Workshop on Rich Representations for Reinforcement Learning

Kurt Driessens
Alan Fern
Martijn van Otterlo

Held in conjunction with the 22^{nd} International Conference on
Machine Learning, August 7, 2005, Bonn, Germany

Overview

Reinforcement learning (RL) has developed into a primary approach to learning control strategies for autonomous agents. The majority of RL work has focused on propositional or attribute-value representations of states and actions, simple temporal models of action, and memoryless policy representations. Many problem domains, however, are not easily represented under these assumptions.

This has led to work that studies the use of richer representations in RL to overcome some of these traditional limitations. This includes for example: relational reinforcement learning, where states, actions, value functions, and policies have relational representations; richer representations of action and policies that incorporate internal state, such as options and MAXQ hierarchies; and the recently introduced predictive state representations where the state of a system is represented in terms of predictive statements about future observations.

The aim of this workshop is to bring together researchers working on various representational aspects of RL to explore their approaches, the relationships among them, their benefits (or drawbacks) for reinforcement learning, and the key representational challenges that remain. Some of the issues/questions we hope to explore in this forum include:

- New algorithms for exploiting rich representations to the fullest. When is it possible to design algorithms for rich representations by reduction to traditional techniques?
- When and how does reinforcement learning benefit from rich representations? Specific real-world successes and failures are of particular interest.
- What is the influence of rich representations on the (re-)usability of reinforcement learning results, or transfer learning (for example through goal parameterization)?
- Should the introduction of rich representations in reinforcement learning be accompanied by different learning goals to keep the learning problems feasible?
- How should we evaluate new algorithms for rich representations? Specific benchmarks that exhibit the weaknesses and benefits of various representational features are of particular interest.

- How can RL benefit from/contribute to existing models and techniques used for (decision-theoretic) planning and agents that already use richer representations, but lack learning?
- Can the interaction between rich representations and the (known and validated) framework of (PO)Markov Decision Processes be characterized in a theoretically rigorous way?

Workshop Summary

The workshop includes twelve submissions from researchers in a variety of representational areas. As such, to promote interaction the workshop will include three invited talks by leading researchers in different representational aspects of RL, each giving a tutorial-style overview and discussion of future research directions. Rich Sutton will cover predictive state representations, which have receive much recent interest. Roni Khardon will cover relational RL (RRL), another recent area with a growing body of work. Ron Parr will cover hierarchical RL (HRL) which is perhaps the most well developed, but still growing, area of the three.

In the area of RRL, Scott Sanner describes an interesting approach based on using relational naive Bayesian networks, showing promising results in the domain of Backgammon. Jan Ramon describes his work on analyzing the convergence of RL based on relational decision trees, which served as one of the first representations used in RRL. In addition, the contribution by Terran Lane and Bill Smart describes a type of relational representation specialized for exploiting spatial structure in RL domains.

In the area of HRL, Victoria Manfredi and Sridhar Mahadevan present an approach that first uses supervised EM learning to acquire a graphical model representation of hierarchical structure, and then uses this structure as a seed for more traditional HRL. Neville Mehta and Prasad Tadepalli introduce a new multi-agent extension to MAXQ, based on sharing subtask value functions across agents. Mehran Asadi and Manfred Huber present an approach to HRL based on discovering subgoals that can then be transferred to related tasks, speeding up learning. Omer Ziv and Nahum Shimkin present a substantially different type of approach for expoiting hierarchical structure—giving a numerical techniques for speeding up value iteration in the presence of a state abstraction hierarchy. Finally, Michael Littman, Carlos Diuk, and Alexander Strehl present a new algorithm MaxQ-Rmax that synergistically combines factored representations, model-based learning, and hierarchies. Interestingly MaxQ-Rmax can be shown to produce near-optimal policies (within the hierarchy) in polynomial time.

Given the widespread use of kernel representations throughout other areas of machine learning, it is somewhat surprising that they have recieved relatively little attention in RL. Yaakov Engel, Shie Mannor, and Ron Meir address this issue and will present their work on the use of Gaussian processes in RL, which facilitates the exploitation of kernels and thus any representation on which a kernel can be defined.

Three contributions explore representational issues that do not fall into the above categories. Ronen Brafman, Guy Shani and Solomon Shimony present an approach to dealing with partial observability based on using U-Tree representations of POMDPs. Kary Framling presents a new bi-memory model that is able to utilize teacher provided heuristic rules in order to more effectively guide exploration. Finally, Seung-Joon Yi and Byoung-Tak present an interesting proposal for merging complex network ("small world" network) theory with reinforcement learning to improve learning rates for large, continuous state space problems.

Table of Contents

Accelerating Action Dependent Hierarchical Reinforcement Learning Through Autonomous Subgoal Discovery

Mehran Asadi ASADI@CSE.UTA.EDU
Manfred Huber HUBER@CSE.UTA.EDU
Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019 USA

Abstract

This paper presents a new method for the autonomous construction of hierarchical action and state representations in reinforcement learning, aimed at accelerating learning and extending the scope of such systems. In this approach, the agent uses information acquired while learning one task to discover subgoals for similar tasks by analyzing the learned policy using Monte Carlo sampling. The agent is able to transfer this knowledge to subsequent tasks and to accelerate learning by creating corresponding subtask policies as abstract actions (options). At the same time, the subgoal actions are used to construct a more abstract state representation using action-dependent state space partitioning, adding a new level to the state space hierarchy. This level serves as the initial representation for new learning tasks. In order to ensure that tasks are learnable, value functions are built simultaneously at different levels of hierarchy and inconsistencies are used to identify actions to be used to refine relevant portions of the abstract state space.

1. Introduction

Autonomous systems are often difficult to program. Reinforcement learning (RL) is an attractive alternative, as it allows the agent to learn behavior on the basis of sparse, delayed reward signals provided only when the agent reaches desired goals. However, standard reinforcement learning methods do not scale well to larger, more complex tasks. One promising approach to scaling is hierarchical reinforcement learning (HRL) (Sutton et al., 1999; Kim & Dean, 2003; Dietterich, 2000; Parr, 1998; Asadi & Huber, 2004).
One of the fundamental steps toward HRL is to automatically establish subgoals. Methods for automatically introducing subgoals have been studied in the context of adaptive production systems, where subgoals are created based on examinations of problem-solving protocols. For RL systems, several researchers have proposed methods by which policies learned for a set of related tasks are examined for commonalities or are probabilistically combined to form new policies. Subgoal discovery has been addressed by several researchers (McGovern & Barto, 2001; Digney, 1996; Drummond, 1997). The most closely related research is that of Digney (Digney, 1996) where states that are visited frequently or states where the reward gradient is high are chosen as subgoals.
The presented work introduces a new method for the autonomous construction of hierarchical actions and state representations in reinforcement learning. In this approach, the agent uses information acquired while learning one task to discover subgoals for similar tasks by analyzing the learned policy using Monte Carlo sampling. The agent is able to transfer knowledge and to accelerate learning of subsequent tasks by creating new subgoals and by off-line learning corresponding subtask policies as abstract actions (options). At the same time, the subgoal actions are used to construct a more abstract state representation using action-dependent state space partitioning. This representation forms a new level in the state space hierarchy and serves as the initial representation for new learning tasks. To ensure that tasks are learnable, value functions are built at different levels of hierarchy and inconsistencies are used to identify actions to be used to refine relevant portions of the abstract state space.

2. Reinforcement Learning

In the RL framework, a learning agent interacts with an environment over a series of time steps $t =$

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

$0, 1, 2, 3, ...$ At each time t, the agent observes the state, s_t , and chooses an action, a_t , which causes a transition to state s_{t+1} and a reward, r_{t+1}. In a Markovian system, the likelihood of the next state and of the reward depend only on the preceding state and the action taken. The objective of the agent is to learn a (possibly probabilistic) mapping from states to actions which maximizes the expected discounted reward received over time. A common RL approach is to approximate the optimal state/action value function, or Q-function, which maps state/action pairs to the maximum expected return starting from the given state and action and thereafter following the best policy.

To permit the construction of a hierarchical learning system, we model our learning problem as a Semi-Markov Decision Problem (SMDP) and use the option framework (Sutton et al., 1999). An option is a temporally extended action which, when selected by the agent, executes until a termination condition is satisfied. While an option is executing, actions are chosen according to the option's own policy. More specifically, an option is a triple $o_i = (I_i, \pi_i, \beta_i)$, where I_i is the set of states in which the option can be initiated; π_i is the option's policy defined over all states in which the option can execute; and β_i is the termination condition defining the probability with which the option terminates in a given state, s. Each option used in this paper bases its policy on its own internal value function. The value of a state s under an SMDP policy π^o is defined as (Boutilier et al., 1999; Sutton et al., 1999):

$$V^{\pi^o}(s) = E[R(s, o_i) + \sum_{s'} F(s'|s, o_i)V^{\pi^o}(s')]$$

where

$$F(s'|s, o_i) = \sum_{k=1}^{\infty} P(s_{t+k} = s'|s_t = s, o_i)\gamma^k \quad (1)$$

and

$$R(s', o_i) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | \epsilon(\pi_i, s, t)] \quad (2)$$

where r_t denotes the reward at time t and $\epsilon(\pi_i, s, t)$ denotes the event of an action under policy π_i being initiated at time t and in state s (Sutton et al., 1999).

3. Autonomous Subgoal Discovery

An example that shows the importance of a subgoal is a room to room navigation task where the agent should discover the utility of doorways. If the agent discovers that a doorway is a subgoal it can learn an option to reach the doorway which, in turn, can accelerate learning of new navigation tasks. The idea of using subgoals is not, however, limited to grid worlds. For example, for a robot arm to pick up an object, an important subtask is the recognition of the object and thus being aware of its presence would be a subgoal.

The main goal of automatic subgoal discovery is to find useful subgoals in the agent's state space. Once they are found, options to those subgoals can be learned and added to the behavioral repertoire of the agent. In the approach presented here, subgoals are identified as states with particular structural properties in the context of a given policy. In particular, we define subgoals as states that, under a given policy, lie on a substantially larger number of paths than would be expected by looking at its successor states. In other words, we are looking for states that form a "funnel" for state space trajectories under the learned policy.

Definition 1 *A state s' is a direct predecessor of state s, if under a learned policy the action in state s' can lead to s i.e., $P(s|s', a) > 0$.*

Definition 2 *The count metric for state s under a learned policy, π, is the sum over all possible state space trajectories weighed by their accumulated likelihood to pass through state s.*

Let $C^*_\pi(s)$ be the count for state s, then:

$$C^1_\pi(s) = \sum_{s' \neq s} P(s|s', \pi(s')) \quad (3)$$

and

$$C^t_\pi(s) = \sum_{s' \neq s} P(s|s', \pi(s'))C^{t-1}_\pi(s') \quad (4)$$

$$C^*_\pi(s) = \sum_{i=1}^{n} C^i_\pi(s) \quad (5)$$

where n is such that $C^n_\pi(s) = C^{n+1}_\pi(s)$ or $n = |S|$. The condition $s' \neq s$ prevents the counting of self loops and $P(s|s', \pi(s'))$ is the probability of reaching state s from state s' by executing action $\pi(s')$. The slope of $C^*_\pi(s_t)$ along a path, ρ, under policy π is:

$$\Delta_\pi(s_t) = C^*_\pi(s_t) - C^*_\pi(s_{t-1}) \quad (6)$$

where s_t is the t^{th} state along the path. In order to identify subgoals, the gradient ratio $\Delta_\pi(s_t)/\max(1, \Delta_\pi(s_{t+1}))$ is computed for states where $\Delta_\pi(s_t) > 0$. A state s_t is considered a potential subgoal candidate if the gradient ratio is greater than a specified threshold $\mu > 1$. Appropriate values for this user-defined threshold depend largely on the characteristics of the state space and result in a number of subgoal candidates that is inversely related to the value of μ. This approach is an extension of the criterion in (Goel & Huber, 2003) with $\max(1, \Delta_\pi(s_{t+1}))$ addressing the effects of potentially obtaining negative gradients due to nondeterministic transitions.

In order to reduce the computational complexity of the above method in large state spaces, the gradient ratio is here computed using Monte Carlo sampling.

Definition 3 *Let $H = \{h_1, ..., h_N\}$ be N sample trajectories induced by policy π, then the sampled count metric, $C^*_H(s)$, for each state s that is on the path of at least one path h_i can be calculated as the average of the accumulated likelihoods of each path, h_i, $1 \leq i \leq N$, rescaled by the total number of possible paths in the environment.*

We can show that for trajectories h_i and sample size N such that

$$N \geq \frac{\max_t C^*_H(s_t)}{\epsilon_N^2} 2(1+\epsilon_N) log(\frac{2}{1-p}) \qquad (7)$$

the following statement is true with probability p:

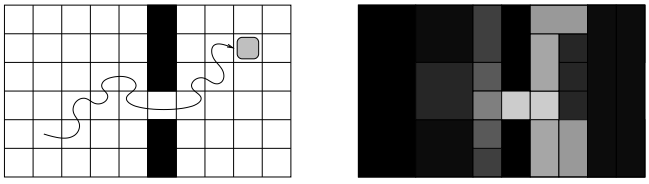
$$|C^*_H(s_t) - C^*_\pi(s_t)| \leq \epsilon_N$$

Theorem 1 *Let $H = \{h_1, ..., h_N\}$ be N sample trajectories induced by policy π with N selected according to Equation 7. If $\frac{\Delta_H(s_t)}{\max(1,\Delta_H(s_{t+1}))} > \mu + \frac{2\epsilon_N(\mu+1)}{\max(1,\Delta_H(s_{t+1}))}$, then $\frac{\Delta_\pi(s_t)}{\max(1,\Delta_\pi(s_{t+1}))} > \mu$ with probability $\geq p$.*

Theorem 1 implies that for a sufficiently large sample size the exhaustive and the sampling method predict the same subgoals with high probability.

3.1. Example

Figure 1(a) shows a two-room example environment on a 10×6 grid. For this experiment, the goal state is placed in the upper right hand portion (gray cell) and each trial is started from the same state in the lower left corner. The action space consists of eight primitive actions (North, East, South, West, Northwest, Northeast, Southwest and Southeast). The world is deterministic and each action succeeds in moving the agent in the chosen direction. With every action the agent receives a negative reward of -1 for a straight action and -1.2 for a diagonal action. In addition, the agent receives a reward of $+10$ when it reaches the goal state. Policy π is learned using Q-learning and the count metric for every state is computed. The agent then evaluates the gradient ratio along the count curve by choosing 10 random trajectories according to π, and picks the states in which the ratio is higher than the specified threshold as subgoal states. Figure 1(b) shows the values of the gradient ratio for each state. In this example, the gradient ratio is less than 4 in all states except the doorway where it is 5.232. The mean of the distribution of gradient ratios over the state space is 3.265 and the standard deviation is 0.208. The threshold is here chosen to be 4 resulting in one subgoal in the location of the doorway. This subgoal could now be used to learn similar tasks in this environment.



(a) (b)

Figure 1. (a) A two room environment with a connecting doorway. The goal is illustrated in gray and the line shows a sample trajectory. (b) Cells are shaded according to the gradient ratio over 10 trajectories with higher ratios indicated by lighter shading.

4. Action Dependent Partitions

Once potential subgoals are discovered, options that terminate in the subgoal states can be learned and added as abstract actions to the action hierarchy available to the agent. In addition, these options can be used to build a more compact representation of the state space. Abstraction is achieved here by partitioning the state space of the original MDP into blocks of states that have similar properties (i.e transition probabilities and reward values).

ϵ-reductions were introduced by Dean et al. (Dean et al., 1997) as a mechanism to derive state space partitions of a MDP that ensure approximately optimal policies to be learned. These partitions depend on the action space and the particular reward function of the task. Kim and Dean, (Kim & Dean, 2003) introduced an algorithm to derive a set of such partitions and used it to learn a policy for the task indicated by the reward function. The resulting policy is ensured to be within an ϵ-dependent quality bound.

Two shortcomings of this algorithm are that it does not address temporally abstract actions and that it requires a complete re-computation of the partitions when a new action is introduced. In addition it requires knowledge of the reward function prior to partitioning, and thus no part of the partitioning transfers across tasks. To address these issues, the algorithm introduced in this section derives partitions with the same approximate optimality properties for the SMDP framework in two phases, the first of which is reward-independent and thus transfers across tasks. This method first constructs the options $o_i = (I_i, \pi_i, \beta_i)$ according to the discovered subgoals. The transition probability function $F(s|s', o_i)$ and the reward function $R(s, o_i)$ can be computed using equations 1 and 2. The transition function can here be completely pre-

computed at the time when the policy itself is learned and as a result, only the discounted reward estimate has to be re-computed for each new learning task.

In the two phase partitioning approach presented here we construct the initial blocks of the partition by distinguishing terminal states (subgoals) for available option from non-terminal states and then refine these blocks based on the transition probability function.

Let $\{s_1, \ldots, s_n\}$ be n discovered subgoals and $\{o_1, \ldots, o_n\}$ be the corresponding options. We construct a partition $P = \{B_1, \ldots, B_n\}$ of state space S such that each set B_i contains all states $s \in I_i$ such that $F(s_i|s, o_i) > 0$, $\beta(s_i) = 1$ and $\cup_{i=1}^{n} B_i = S$.

Definition 4 *A partition $P = \{B_1, \ldots, B_n\}$ of the state space of an MDP has approximate stochastic bisimulation homogeneity if and only if for each $B_i, B_j \in P$ and for each $s, s' \in B_i$:*

$$\left| \sum_{s'' \in B_j} F(s''|s, o_i(s)) - \sum_{s'' \in B_j} F(s''|s', o_i(s')) \right| \leq \delta \quad (8)$$

where $0 \leq \delta \leq 1$.

We say that a block B_i is δ-stable with respect to block B_j if and only if Equation 8 holds. B_i is δ-stable if B_i is δ-stable with respect to all blocks of P.

To form partitions, each block is checked for δ-stability and unstable blocks are split until no unstable blocks remain. When a block B_k is found to be unstable with respect to block B_l, we replace B_k by a set of sub-blocks $B_{k_1}, \ldots, B_{k_m}$ such that B_{k_i} is maximal sub-block of B_k that is δ-stable with respect to B_l. To facilitate modifications in the action space, this process is first performed for each option individually. The blocks of the final partition are then formed by intersecting all blocks for each o_i that are used followed by a refining stage that achieves δ-stability for the intersections (Asadi & Huber, 2004). This reduces the overhead required when the action set changes to the intersection and the final refinement step.

If the reward structure becomes available, the second phase of the partitioning technique further refines the partition with the following reward criterion:

$$|R(s, o_i) - R(s', o_i)| \leq \epsilon \quad (9)$$

Given a particular subset of options, an appropriate abstract state space representation for the learning task can thus be derived which is stable according to criteria in Equations 8 and 9. Furthermore, representation changes due to changes in the action set can be performed efficiently and a simple mechanism can be provided to use the previously learned value function as a starting point when such representation changes occur. This is particularly important if actions are added over time to permit refinement of the initially learned policy by permitting finer-grained decisions.

5. Learning Method

Let $P = \{B_1, \ldots, B_n\}$ be a partition for state space S derived by the action-dependent partitioning method, using subgoals $\{s_1, \ldots, s_k\}$ and options to these subgoals $\{o_1, \ldots, o_k\}$. If the goal state G belongs to the set of subgoals $\{s_1, \ldots, s_k\}$, then G is achievable by options $\{o_1, \ldots, o_k\}$ and the task is learnable according to Theorem 3. However, if $G \notin \{s_1, \ldots, s_k\}$ then the task may not be solvable using only the options that terminate at subgoals. The proposed approach solves this problem by maintaining a separate value function for the original state space while learning a new task on the partition space derived from only the subgoal options. During learning, the agent has access to the original actions as well as all options, but makes decisions only based on the abstract partition space information.

While the agent tries to solve the task on the abstract partition space, it computes the difference in Q-values between the best actions in the current state in the abstract state space and in the original state space. If the difference is larger than a constant value (given by Theorem 2), then there is a significant difference between different states underlying the particular block that was not captured by the subgoal options. Theorem 2 (Kim & Dean, 2003) shows that if blocks are stable with respect to all actions the difference between the Q-values in the partition space and in the original state space is bounded by a constant value.

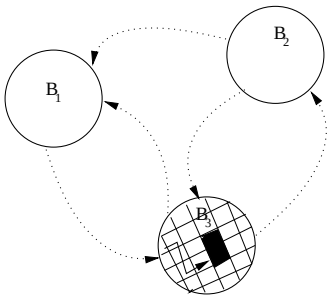


Figure 2. An abstract state space with 3 blocks (B_1, B_2, B_3). Options are shown using dotted curves and original action are illustrated with solid lines. The black cell in block B_3 is the goal state. Since the Q-value in block B_3 is significantly smaller than the one of the underlying goal state this block is refined using the options and the primitive actions.

Theorem 2 *Given an MDP $M = (S, A, T, R)$ and a partition P of the state space M_P , the optimal value*

function of M given as V^* and the optimal value function of M_P given as V_P^* satisfy the bound on the distance

$$\| V^* - V_P^* \|_\infty \leq 2\left(1 + \frac{\gamma}{1-\gamma}\epsilon_p\right)$$

where $\epsilon_p = \min_{V_p} \| V^* - V_P^* \|_\infty$ *and*

$$LV(s) = \max_a [R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s)]$$

When the difference between the Q-values for states in block B_i are greater than $2(1+\frac{\gamma}{1-\gamma}\epsilon_p)$, then the primitive action that achieves the highest Q-value on the original state in the MDP will be added to the action space of those states that are in block B_i and block B_i is refined until it is stable for the new action set. Once no such significant difference exists, the goal will be achievable in the resulting state space according to Theorem 3. This procedure is illustrated in Figure 2.

Theorem 3 *For any policy π for which the goal G can be represented as a conjunction of terminal sets (subgoals) of the available actions in the original MDP M, there is a policy π_P in the reduced MDP, M_P , that achieves G as long as for each state s_t in M for which there exists a path to G , there exists a path such that $F(G|s_t, \pi_P(s_t)) > \delta$.*

6. Experimental Results

The main goal of this experiment is to demonstrate the potential of the proposed approach to hierarchical learning in accelerating learning in a stochastic environment. The task of the agent is to find an object in a randomly chosen cell, to pick it up and drop it in another randomly chosen location. Figure 3 illustrates the environment for this experiment. The state space consists of three grid worlds that are connected through stair ways (arrows). The dark cells represent obstacles and the actions are GoNorth, GoEast, GoSouth, GoWest, GoUp, GoDown, OpenArm, CloseArm, Pickup and Drop. Actions for stairways are defined as sequences of length 10 of GoUp or GoDown. The cost for each single step action is -1 and each action for navigation succeeds with probability 0.5 while leading to either side with probability 0.25. Actions OpenArm, CloseArm, Pickup and Drop always succeed. The reward in the goal state is 100.

To demonstrate the power of hierarchical learning, the agent is first given the task of learning a policy to move from a fixed starting location to a particular goal point. It then uses this policy to extract subgoals by generating random samples according to the learned policy. The samples are paths of length 40 and the subgoals are discovered as described in Section 3. Figure 4(b) illustrates the number of subgoals that are discovered by Monte Carlo sampling. As illustrated in Figure 4(b), the total number of samples needed to learn almost all of the subgoals is approximately 127. Figure 4(a) shows that the total time for subgoal using 127 samples is less that 30 seconds, which is 5 times faster than using the entire state space. The extracted subgoals in this experiment consist of a set of doorways, entry points of stairways and states where the agent acquired the object.

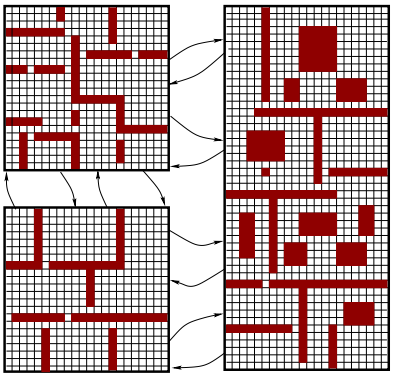


Figure 3. A three room environment connected by stairways (arrows). Black cells indicate obstacles. The task for the agent is to navigate this environment, find an object, pick it up, move it to another locations, and drop it.

Once subgoals are extracted, options for these subgoals are derived and the agent is given the final learning task described in the beginning of the section. Figure 5 shows the learning speed and quality of the learned policies with and without refinement of abstract states based on Q-value inconsistencies. This experiment shows that while the goal state is here not reachable using only the subgoal actions, almost equal performance to the original MDP is achieved with refinement based on primitive actions. Furthermore, learning on the partition space with on-line refinement of partitions according to Q-value differences in the abstract and original state representations significantly outperforms both learning on an a priori refined representation and on the original state space.

7. Conclusion

This paper presents an efficient method for autonomously constructing a hierarchical state and action space for SMDPs. To do this, it first discovers subgoals by analyzing previously learned policies for states with particular structural properties. Once subgoals are derived, it learns options to achieve these subgoals off-line and includes these into the action hierarchy available to the agent. Using the subgoal options, it then uses action-dependent state space partitioning

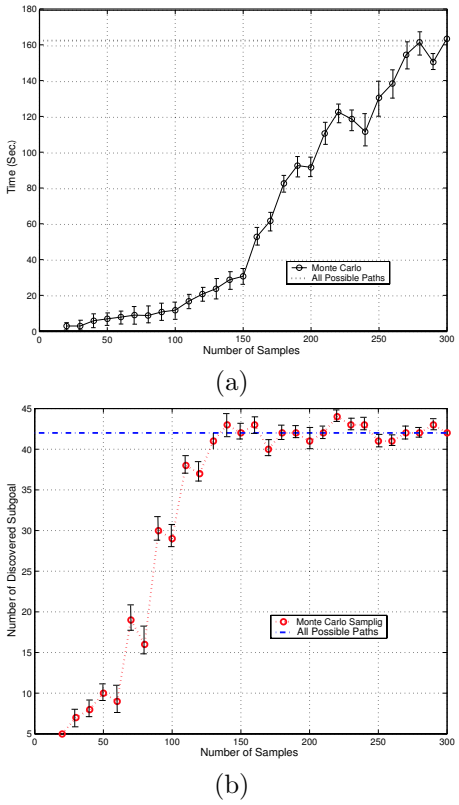


Figure 4. (a) Comparison between the run times for subgoal discovery using the entire state space and Monte Carlo sampling. (b) Number of samples that are needed to discover all useful subgoals.

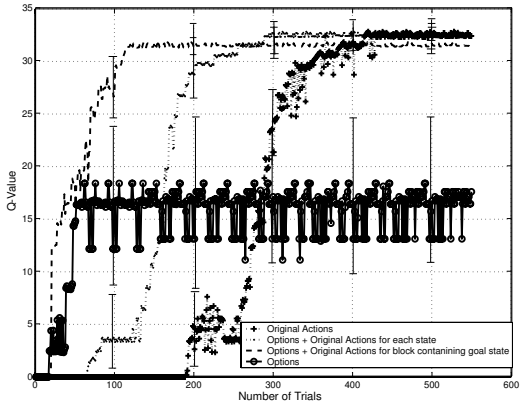


Figure 5. Comparison of policies derived on the partition and the original state space. The policies using the options to subgoals on the partition space with primitive action refinement converge significantly faster than the optimal policy in the original state space.

to derive an abstract state space. Learning of subsequent tasks is addressed on the abstract state space. To ensure that the new task is learnable, the system maintains a separate value function for the original state space and refines the representation when significant inconsistencies between this value function and the one derived on the abstract partition space are detected. Experiments using this approach show that this approach significantly accelerates learning of even complex task strategies while maintaining the quality of the found solution within predetermined bounds.

References

Asadi, M., & Huber, M. (2004). State Space Reduction For Hierarchical Reinforcement Learning. *Proc. FLAIRS* (pp. 509–514).

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *AI Research*, *11*, 1–94.

Dean, T., Givan, R., & Leach, S. (1997). Model Reduction Techniques for Computing Approximately Optimal Solutions for Markov Decision Processes. *Proc. UAI* (pp. 124–131).

Dietterich, T. G. (2000). An Overview of MAXQ Hierarchical Reinforcement Learning. *Lecture Notes in Computer Science*, *1864*, 26–44.

Digney, B. (1996). Emergent hierarchical control structures: Learning reactive / hierarchical relationships in reinforcement environments. *Proc. Conf. Simulation of Adaptive Behavior* (pp. 363–372).

Drummond, C. (1997). Using a Case Base of Surfaces to Speed-Up Reinforcement Learning. *Proc. Int. Conf. Case-Based Reasoning* (pp. 435–444).

Goel, S., & Huber, M. (2003). Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies. *Proc. FLAIRS* (pp. 346–350).

Kim, K., & Dean, T. (2003). Solving Factored MDPs using Non-Homogeneous Partitions. *Artificial Intelligence*, *147*, 225–251.

McGovern, A., & Barto, A. (2001). Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. *Proc. ICML* (pp. 361–368).

Parr, R. (1998). *Hierarchical Control and Learning for Markov Decision Processes*. Doctoral dissertation, University of California, Berkeley, CA.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and Semi-MDPs: Learning, Planning, and Representing Knowledge at Multiple Temporal Scales. *Artificial Intelligence*, *112*, 181–211.

Partial Observability Under Noisy Sensors — From Model-Free to Model-Based

Ronen I. Brafman BRAFMAN@CS.BGU.AC.IL
Guy Shani SHANIGU@CS.BGU.AC.IL
Solomon E. Shimony SHIMONY@CS.BGU.AC.IL
Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel

Abstract

Agents learning to act in a partially observable domain may need to overcome the problem of noisy output from the agent's sensors. Research in the area has focused on model-free methods — methods that learn a policy without learning a model of the world. When the agent's sensors provide deterministic output, model-free methods produce close to optimal results. However, when the noise in the sensors increases, these methods provide less accurate policies. Another, less explored, option is the model-based approach — learning a POMDP model of the world, and obtaining an optimal policy from the learned model. In this paper we show the superiority of model-based techniques over model-free methods in the presence noisy sensors. We demonstrate how two important model-free algorithms: internal memory, and Utile Suffix Memory, can be used to learn a POMDP model of the environment.

1. Introduction

Reinforcement learning (RL) in partially observable domains (Cassandra et al., 1994) can take one of two forms; the agent can learn a policy directly, or it can learn a model of the environment, usually represented as a Partially Observable Markov Decision Process (POMDP) [1] , and solve it (Chrisman, 1992; Nikovski, 2002). This approach has not been favored by researchers, as learning a model appears to be a difficult task, and computing an optimal solution may prove impractical for large models. However, in the past few years, much progress in the area of approximate (Poupart & Boutilier, 2004; Spaan & Vlassis, 2004) solutions for POMDP models has been made. In view of these advances, we reconsider the model-based methods, focusing on their ability to handle sensor noise.

Identifying the "real", hidden, states of the world using noisy sensors is a difficult task. Even when the agent's sensors provide it with deterministic output, the agent may still suffer from the problem of *perceptual aliasing* (Chrisman, 1992), when different actions should be executed in two states where sensors provide the same output. For example, in Figure 1(a) the left and right corridors are perceptually aliased if sensors can only sense adjacent walls. While disambiguating the state space may require the agent to remember events that happen arbitrarily far in the past, we focus our attention on the class of problems that can be solved looking back a finite number of steps. Model-free[2] methods - such as augmenting the observations with internal memory (Peshkin et al., 1999), or using variant-length finite history windows (McCallum, 1996) - can be used to disambiguate the perceptually aliased states for such problems. When the agent's sensors provide deterministic output, learning to properly identify the underlying world states reduces the problem to a fully observable MDP, making it possible for methods such as Q-learning to compute an optimal policy.

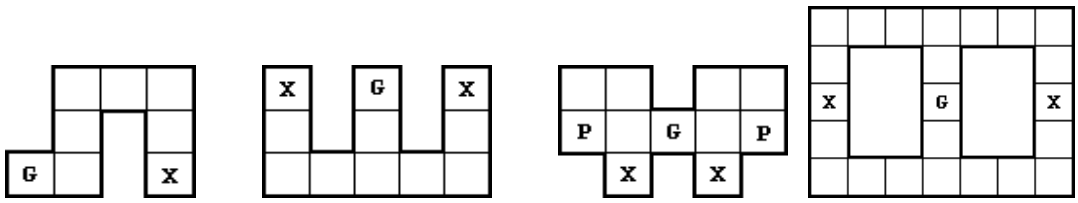
When sensors provide slightly noisy output, model-free methods still produce close to optimal results, but as noise in the sensors increases, their performance rapidly decreases (Shani & Brafman, 2004). This is because disambiguating the perceptually aliased states under noisy sensors does not result in an MDP, but rather in a POMDP. POMDP models are harder to solve, but their solution handles noisy observations optimally.

Using model-free methods to create a POMDP has been previously suggested by Nikovski (Nikovski, 2002) in his

[1] See Section 2.1 for an overview of MDPs and POMDPs

[2] As the discussed problems are properly defined as a POMDP, we call methods that do not learn all the POMDP parameters "model-free", though they may learn state representations.

Appearing in *Proceedings of the 22^{st} International Conference on Machine Learning*, Bonn, Germany, 2005.



(a) (b) (c) (d)

Figure 1. Four maze domains. The agent receives a reward of 9 upon reaching the goal state (marked with 'G'). Immediately afterwards (in the same transition) the agent is transferred to one of the 'X' states. Arrival at a 'P' state results in a penalty (negative reward) of 9.

PhD dissertation. Nikovski compared his method, based on McCallum's earlier NSM algorithm to other state merging techniques and to classical POMDP learning methods based on the Baum-Welch algorithm, demonstrating how classical approaches fail to converge on even small domains. Nikovski did not show his models to be superior to any model-free techniques and did not experiment with sensor noise.

In this paper, we show how both McCallum's Utile Suffix Memory (USM) algorithm, that learns a variant-length finite history window, and the internal memory approach suggested by Peshkin *et al.* can be used to initialize a POMDP model of the world. We continue to solve the resulting models and compare the average reward collected by model-free and model-based approaches. We then show that the resulting models provide superior results to the policy learned by the model-free methods. This indicates that the use of an explicit model is advantageous for diverse methods of initializing the model.

We note that using an explicit model of the environment can have many other advantages such as taking the "value of information" into consideration, and helping to find the areas of the world that require learning, and may hold potential rewards, but we leave those topics to future research.

The main contribution of this paper is in showing that a solution for a POMDP learned from a model-free method provides improved performance over the original method. Another contribution is the development of the specific techniques that apply this idea to the memory bits and the USM schemes. Empirical results for the above memory schemes show the advantages using the derived POMDP in these cases.

This paper is structured as follows: we begin (Section 2) with an overview over MDPs, POMDPs, the memory bits and USM schemes. We then explain how the learned policy of model-free methods can be used to construct a POMDP in Section 3. We provide an experimental evaluation of our work in Section 4, followed by a short discussion in Section 5 and conclude in Section 6.

2. Background

2.1. MDPs and POMDPs

A Markov Decision Process (MDP) (Howard, 1960) is a model for sequential stochastic decision problems. An MDP is a four-tuple: $\langle S, A, R, tr \rangle$, where S is the set of the states of the world, A is a set of actions an agent can use, R is a reward function, and tr is the stochastic state-transition function. A solution to a MDP is a policy $\pi : S \rightarrow A$ that defines which action should be executed in each state.

Various exact and approximate algorithms exist for computing an optimal policy, and the best known are policy-iteration (Howard, 1960) and value-iteration (Bellman, 1962). Solving MDPs is known to be a polynomial problem in the number of states, and therefore exponential in the number of state variables.

A well known extension to the MDP model is the Partially Observable Markov Decision Process (POMDP) model (Cassandra et al., 1994). A POMDP is a six-tuple $\langle S, A, R, tr, \Omega, O \rangle$, where S, A, R, tr define an MDP, Ω is a set of possible observations and $O(a, s, o)$ is the probability of executing action a, reaching state s and observing o. In a POMDP the agent is unable to identify the current state and is therefore forced to estimate the current state given the set of current observations (e.g. output of the robot sensors). In most applications a POMDP is more natural and complete formalization than an MDP, but using POMDPs increase the difficulty of computing an optimal solution.

Solving a POMDP is an extremely difficult computational problem, and various attempts have been made to compute approximate solution problems that work reasonably well in practice, such as randomized point based value iteration (Spaan & Vlassis, 2004).

The idea of learning a POMDP model of the environment was examined by early researchers (Chrisman, 1992) who used a variant of the Baum-Welch algorithm for learning hidden Markov models, refining the state space when it was observed to be inadequate. These methods were slow to converge and could not outperform the rapid convergence and reasonable results generated by model-

free methods. Nikovski (Nikovski, 2002) used McCallum's earlier model-free method, Nearest Sequence Memory (NSM), to identify the states of the world and learn the transition, reward, and observation functions. He showed that the learned models produced superior results to the models obtained by using the Baum-Welch algorithm. His models, however, were tested on domains with little noise, and are much less adequate when sensors are noisy. This is to be expected, as NSM handles noisy observations poorly, while USM can still produce reasonable results, though in no way optimal.

Nikovski also did not attempt to use any modern technique for solving his models and obtaining a policy. Instead, he experimented with approximate methods based on the solution to the underlying MDP model such as Q_{MDP}.

2.2. Memory bits

Early research in model-free techniques has shown that MDP based techniques such as Q-learning, SARSA and eligibility traces (Sutton & Barto, 1998) fail to converge in the presence of perceptual aliasing.

Peshkin *et al.* has suggested to augment the agent state space with bits of internal memory (though he referred to it as external), and actions that change the value of a single memory bit. The agent can therefore choose to either execute an action that influences the environment, or flip one of its internal memory bits. Q values are learned for all such actions using any RL technique, such as SARSA(λ) — SARSA with eligibility traces. Agents with internal memory can learn to remember events that happened arbitrarily far in the past in order to disambiguate the perceptual aliasing.

2.3. Utile Suffix Memory

Instance-based state identification (McCallum, 1996) resolves perceptual aliasing with variable length short term memory. An instance is a tuple $T_t = \langle T_{t-1}, a_{t-1}, o_t, r_t \rangle$ — the individual observed raw experience. Algorithms of this family keep all the observed raw data (sequences of instances), and use it to identify matching subsequences. The algorithm assumes that if the suffix of two sequences is similar both were likely generated in the same world state.

Utile Suffix Memory creates a tree structure, based on the well known suffix trees for string operations. This tree maintains the raw experiences and identifies matching suffixes. The root of the tree is an unlabeled node, holding all available instances. Each immediate child of the root is labeled with one of the observations encountered during the test. Nodes at the second level are labeled by actions, and so forth.

Inserted instances are split based on their latest observation o_t. At the next level, instances are split based on the last action of the instance a_t. Then, we split again based on (the next to last) observation o_{t-1}, etc. All nodes act as buckets, grouping together instances that have matching history suffixes of a certain length. Leaves take the role of states, holding Q-values and updating them. The deeper a leaf is in the tree, the more history the instances in this leaf share.

Leaves should be split if their descendants show a statistical difference in expected future discounted reward associated with the same action. We split a node if knowing where the agent came from helps predict future discounted rewards. Thus, the tree must keep what McCallum calls fringes, i.e., subtrees below the "official" leaves.

We define the expected discounted reward of instance T_i:

$$Q(T_i) = r_i + \gamma U(L(T_{i+1})) \quad (1)$$

where $L(T_i)$ is the leaf associated with instance T_i and $U(s) = max_a(Q(s,a))$.

After inserting new instances into the tree, we update Q-values in the leaves using:

$$R(s,a) = \frac{\sum_{T_i \in T(s,a)} r_i}{|T(s,a)|} \quad (2)$$

$$Pr(s'|s,a) = \frac{|\forall T_i \in T(s,a), L(T_{i+1}) = s'|}{|T(s,a)|} \quad (3)$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} Pr(s'|s,a)U(s') \quad (4)$$

We use s and s' to denote the leaves of the tree, as in an optimal tree configuration for a problem the leaves of the tree define the sates of the underlying MDP. The above equations therefore correspond to a single step of the value iteration algorithm used in MDPs.

Now that the Q-values have been updated, the agent chooses the next action to perform based on the Q-values in the leaf corresponding to the current instance T_t:

$$a_{t+1} = argmax_a Q(L(T_t), a) \quad (5)$$

McCallum uses the fringes of the tree for a smart exploration strategy. In our implementation we use a simple ϵ-greedy technique for exploration.

3. Constructing Models from Model-Free Methods

Any model-free technique designed to learn deterministic policies in a partially observable domain with perceptual aliasing, must employ some type of internal memory. The structure of the internal memory can be used to initialize a POMDP model. The method for converting the internal

memory into depends on the way the model-free method constructs its memory state and transitions.

Model-free methods have been extensively studied, and there are many other approaches to resolving perceptual aliasing we have not reviewed, including the use of finite-state automata (FSA) (Meuleau et al., 1999), which can be viewed as a special case of the memory-bits approach, but can learn faster and more accurately and the use of neural networks for internal memory (Lin & Mitchell, 1992; Hochreiter & Schmidhuber, 1997). It is likely that those approaches can also be used to initialize a POMDP model, similarly to USM and memory bits. We note that most researchers test their algorithms on environments with very little noise, and do not analyze the effect of noisy sensors.

3.1. Creating a POMDP from Utile Suffix Memory

After the USM algorithm has generated a tree structure, one can use this tree structure to create a POMDP. The state space is defined as the set of leaves computed by USM. We note that this state representation is not necessarily compact, as it is quite possible that if a perceptually aliased state can be reached from two different locations, it may have two different leaves that represent it.

Obtaining the POMDP parameters from the USM tree structure is straightforward. The actions (A) and observations (Ω) are known to the agent prior to learning the model. As stated above, the leaves of the tree (after convergence of the USM algorithm) are used as the states of S. The transition function ($tr(s, a, s')$) is defined by Equation 3 and the reward function ($R(s, a)$) by Equation 2.

We derive the observation function for the POMDP from a sensor model. In many domains, it is natural to have some idea about how observations relate to features of the real world. For example, in robotics, we usually have a reasonable idea of what is the probability that an object exists in front of us if a sonar or a laser distance sensor indicates this. Thus, in our work we assume the existence of such a model. We note that we can learn an observation function of the form $pr(o|a_t, s_{t-1})$, where action a_t is executed in state s_{t-1}. This is useful when we have pure sensing actions. Currently, we are not able to learn a sensor model of the form $pr(o|a_t, s_t)$, where s_t is the state resulting from the execution of action a_t.

3.2. Creating a POMDP from Memory Bits

Once the memory bits algorithm has run for a while, one can use the learned Q-table and the observed instances to initialize the POMDP state space. Let us define $s = \langle o, m \rangle$ the agent state composed of the sensor observation o and the agent internal memory state m. An observation o originates in a perceptually aliased state if there exists two agent states $s =< o, m >$, $s' =< o, m' >$, such that $m \neq m'$ and $max_a Q(s, a) \neq max_a Q(s', a)$, and none of these actions is an action that flips a memory bit. In other words, state are perceptually aliased if the agent learned that it needs to act differently observing the same sensor output, but different internal memory states.

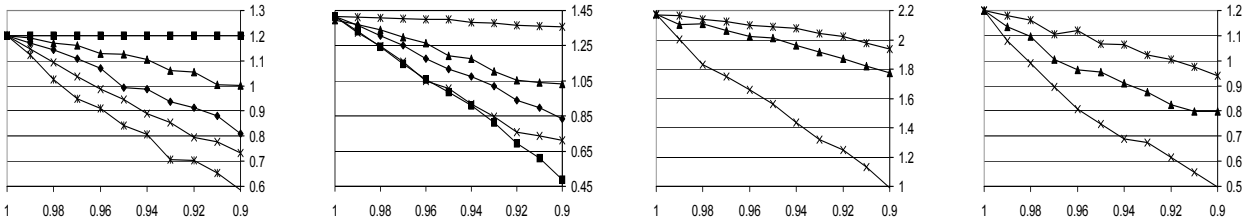
We can now merge every $s =< o, m >$ and $s' =< o, m' >$ that are not perceptually aliased. Using the observed instances, the transition $tr(s, a, s')$ and reward $R(s, a)$ functions can be computed, much the same way as we did for the USM based model. Again, we assume that the observation function is pre-defined.

4. Experimental Results

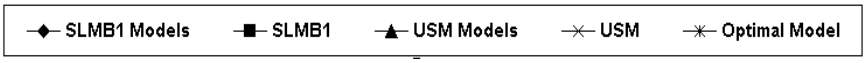
In our experiments we ran the USM-based POMDP on the toy mazes in Figure 1. While these environments are uncomplicated compared to real world problems, they demonstrate important problem features such as multiple perceptual aliasing (Figure 1(b)) and the need for an information gain action (Figure 1(c)). While USM is limited in scaling up to real-world problems, its successor, U-Tree, handles larger domains, and we note that all our methods can be implemented on U-Tree much the same way as for USM.

We ran both USM and SARSA(λ) with one additional memory bit on the mazes in Figure 1. Once the average reward collected by the algorithms passed a certain threshold, their current state (USM's tree structure, and SARSA's Q-table) was kept, exploration was stopped (as the POMDP policy does not explore). In all tests, the convergence of USM was much faster (about four times faster) than the corresponding memory-bits method. Then, the runs were continued for 5000 iterations to calculate the average reward gained by the converged algorithm. The agent current state was then used to learn a POMDP model as explained above. The model was solved by the Perseus algorithm (Spaan & Vlassis, 2004), and the resulting model was executed for another 5000 iterations. This process was repeated 10 times for each point in our graphs, and the presented results are an average of these executions. To show the optimal possible policy, we manually defined a POMDP model for each of the mazes above, solved it using Perseus and ran the resulting policy for 5000 iterations. This was done only once, as there is no learning process involved.

The agent in our experiments has four sensors allowing it to sense an adjacent walls. Sensors have a boolean output with probability α of being accurate. The probability of all sensors providing the correct output is therefore α^4. Upon receiving a reward or punishment, the agent is transformed to any of the states marked by X. If the agent bumps into a wall it pays a cost (a negative reward) of 1. For every move

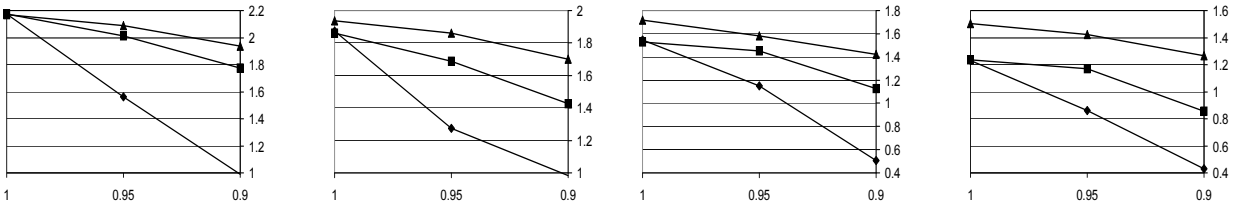


(a) Results for Figure 1(a) (b) Results for Figure 1(b) (c) Results for Figure 1(c) (d) Results for Figure 1(d)

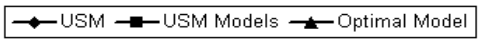


(e) Legend for the above graphs

Figure 2. Results for the mazes in Figure 1. In all the above graphs, the X axis contains the diminishing sensor accuracy, and the Y axis marks average reward per agent action. The above results are averaged over 10 different executions for each observation accuracy and method. All variances were below 0.015 and in most cases below 0.005.



(a) Deterministic actions (b) 0.1 action noise (c) 0.2 action noise (d) 0.3 action noise



(e) Legend for the above graphs

Figure 3. Results for the maze in Figure 1(c) using various levels of action noise (i.e., (1 – action-success-probability)). In all the above graphs, the X axis is the diminishing sensor accuracy, and the Y axis is the average reward per agent action. The above results are averaged over 10 different executions for each observation accuracy and method. All variances were below 0.015 and in most cases below 0.005.

the agent pays a cost of 0.1.

Figure 4 shows the average collected reward for each method when α (the sensor accuracy) varies from 1.0 (deterministic sensor output) to 0.9 (probability 0.65 for detecting all features correctly). SLMB1 stands for adding a single memory bit to the Sarsa(λ) algorithm. SLMB1 Model and USM Model are the output of executing the memory bits and the USM algorithms, respectively, creating a POMDP from the algorithm output, solving it and executing the resulting policy. Optimal Model is the manually defined POMDP model. In the two latter mazes, the memory bits algorithm needed two memory bits and failed to converge as sensor noise increased. We report results only for USM and the manually defined POMDP on these domains.

As seen from the results, the model-based methods greatly improve the original model-based techniques and the advantage becomes more important as noise in the sensors increases. The memory bits based model does not perform as well as the USM based model, probably due to the inaccurate model parameters that were learned because of the inability to explore all states and actions. The USM model in our experiments is also suboptimal, mainly because several leaves correspond to the same world state.

In most cases actions in MDPs and POMDPs do not have deterministic effects. It is quite possible that an action attempted by an agent can fail (in our experiments, a failed action leaves the agent in the same state). Figure 4 shows the results of decreasing action success probability. While the model computed from the USM tree becomes farther from the optimal model, it still outperforms USM by approximately the same amount. The USM based models performance degrades partially due to the increased number of leaves (and hence, states) in the presence of noisy actions, as failed transitions cannot be expressed in a single leaf and instead deeper branches are created for such histories.

The memory bits model in our tests produced smaller state spaces then USM. This is because the memory bits algorithm defines a constant upper bound on the number of states defined by all the possible combinations of the internal memory states and the observations.

5. Discussion

As we have seen above, the performance of model-free techniques degrades when sensor noise increases. This is due to the application of methods designed for learning in a fully observable MDP on a POMDP. These methods assume that the agent knows at each point in time its exact location. When the agent sensors are accurate, resolving the perceptual aliasing indeed results in an MDP and the model-free methods produce reasonable results. In the presence of low sensor noise, the model-free algorithms still produce reasonable approximations, but as sensor noise increases the agent often does not properly estimates its current state resulting in both the execution of wrong actions, and with an unreliable value function.

POMDPs can be accurately solved using the belief state MDP, but the current model-free methods have no concept of a belief state or the belief state MDP. Defining a POMDP, solving it, and maintaining a belief state will therefore show superior results to using any model-free method.

Splitting the learning process into model learning and afterwards policy computation is undesirable. It would be best to combine these steps together, and to create an online learning application that computes a POMDP model together with a policy. We explore this approach and demonstrate that the policy learned by the online learning algorithm improves the original USM performance (Shani et al., 2005).

6. Conclusions and Future Work

In this paper we explored the advantages of model-based methods over model-free methods for acting in partially observable domains with noisy sensors. As learning the model is difficult, we have shown how an agent can execute a model-free method until it converges and then use the learned data to initialize a POMDP model that outperforms the original model-free method.

Future research should focus on other advantages of model-based techniques, such as their ability to consider the "value of information". The current USM algorithm we rely upon creates sub-optimal state spaces, as it may create many different states that correspond to different paths for arriving at the same "real" state. In the future we intend to look for ways to improve USM to create less leaves, such as modifying the current tree structure into a DAG. This will enable a POMDP learning mechanism that can scale up to full sized applications.

Acknowledgments

Partially supported by the Paul Ivanier Center for Robotics and Production Management.

References

Bellman, R. E. (1962). *Dynamic programming*. Princeton University Press.

Cassandra, A. R., Kaelbling, L. P., & Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. *AAAI'94* (pp. 1023–1028).

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *AAAI'02* (pp. 183–188).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*, 1735–1780.

Howard, R. A. (1960). *Dynamic programming and markov processes*. MIT Press.

Lin, L.-J., & Mitchell, T. M. (1992). *Memory approaches to reinforcement learning in non-markovian domains* (Technical Report CMU-CS-92-138).

McCallum, A. K. (1996). *Reinforcement learning with selective perception and hidden state*. Doctoral dissertation, University of Rochester.

Meuleau, N., Peshkin, L., Kim, K., & Kaelbling, L. P. (1999). Learning finite-state controllers for partially observable environments. *UAI'99* (pp. 427–436).

Nikovski, D. (2002). *State-aggregation algorithms for learning probabilistic models for robot control*. Doctoral dissertation, Carnegie Mellon University.

Peshkin, L., Meuleau, N., & Kaelbling, L. P. (1999). Learning policies with external memory. *ICML'99* (pp. 307–314).

Poupart, P., & Boutilier, C. (2004). VDCBPI: an approximate scalable algorithm for large POMDPs. *NIPS 17*. MIT Press.

Shani, G., & Brafman, R. I. (2004). Resolving perceptual aliasing in the presence of noisy sensors. *NIPS'17*.

Shani, G., Brafman, R. I., & Shimony, S. E. (2005). Model-based online learning of POMDPs. *BISFAI'05*.

Spaan, M. T. J., & Vlassis, N. (2004). *Perseus: Randomized point-based value iteration for POMDPs* (Technical Report IAS-UVA-04-02). University of Amsterdam.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Reinforcement learning with kernels and Gaussian processes

Yaakov Engel YAKI@CS.UALBERTA.CA
Dept. of Computing Science, University of Alberta, Edmonton, Canada

Shie Mannor SHIE@ECE.MCGILL.CA
Dept. of Electrical and Computer Engineering, McGill University, Montreal, Canada

Ron Meir RMEIR@EE.TECHNION.AC.IL
Dept. of Electrical Engineering, Technion Institute of Technology, Haifa 32000, Israel

Abstract

Kernel methods have become popular in many sub-fields of machine learning, with the exception of reinforcement learning; they facilitate rich representations, and enable machine learning techniques to work in diverse input spaces. We describe a principled approach to the policy evaluation problem of reinforcement learning. We present a temporal difference (TD) learning algorithm using kernel functions. Our approach allows the TD algorithm to work in arbitrary spaces as long as a kernel function is defined in this space. This kernel function is used to measure similarity between states. The value function is described as a Gaussian process and we obtain a Bayesian solution by solving a generative model. A SARSA based extension of the kernel-based TD algorithm is also mentioned.

1. Introduction

In Engel et al. (2003) the use of Gaussian Processes (GPs) for solving the Reinforcement Learning (RL) problem of value estimation was introduced. Since GPs belong to the family of kernel machines, they bring into RL the high, and quickly growing representational flexibility of kernel based representations, allowing them to deal with almost any conceivable object of interest, from text documents and DNA sequence data to probability distributions, trees and graphs, to mention just a few (see Shawe-Taylor & Cristianini, 2004, and references therein). Moreover, the use of Bayesian reasoning with GPs allows one to obtain not only value estimates, but also estimates of the *uncertainty* in the value, and this in large and even infinite MDPs.

In this extended abstract we present our approach concerning learning with Gaussian processes and kernel methods. We show how to use GPs and kernels to perform TD algorithms on any input space where a kernel function can be defined. The results reported here are based on Engel et al. (2003); Engel et al. (2005); Engel and Mannor (2005) as well as on some ongoing work.

We start by providing a model to the value function based on the discounted return in Section 2. We then describe an online implementation in Section 3. A SARSA based algorithm is briefly mentioned in Section 4. A short summary follows in Section 5.

2. Modeling the Value Via the Discounted Return

A fundamental entity that is of interest in RL is the *discounted return*. Much of the RL literature is concerned with the expected value of this random process, known as the *value function*. This is mainly due to the simplicity of the Bellman equations which govern the behavior of the value function, and because of two provably convergent algorithms (of which many variations exist) that arise from Bellman's equations – value iteration and policy iteration. However, some valuable insights may be gained by considering the discounted return directly and its relation with the value.

A Markov Decision Process (MDP) is a tuple $(\mathcal{X},\mathcal{U},R,p)$ where $\mathcal{X}$ and $\mathcal{U}$ are the state and action spaces, respectively; $R : \mathcal{X} \to \mathbb{R}$ is the immediate reward, which may be random, in which case $q(\cdot|\mathbf{x})$

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

denotes the distribution of rewards at the state $\mathbf{x}$; and $p : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to [0,1]$ is the transition distribution, which we assume is stationary. Note that we do not assume that $\mathcal{X}$ is Euclidean, or that it is even a vector space. Instead, we will assume that a kernel function, k, is defined on $\mathcal{X}$. A *stationary policy* $\mu : \mathcal{X} \times \mathcal{U} \to [0,1]$ is a mapping from states to action selection probabilities. Given a fixed policy μ, the transition probabilities of the MDP are given by the *policy-dependent state transition probability distribution* $p^\mu(\mathbf{x}'|\mathbf{x}) = \int_{\mathcal{U}} d\mathbf{u} p(\mathbf{x}'|\mathbf{u},\mathbf{x})\mu(\mathbf{u}|\mathbf{x})$. The *discounted return* $D(\mathbf{x})$ for a state $\mathbf{x}$ is a random process defined by

$$D(\mathbf{x}) = \sum_{i=0}^{\infty} \gamma^i R(\mathbf{x}_i)|\mathbf{x}_0 = \mathbf{x}, \text{ with } \mathbf{x}_{i+1} \sim p^\mu(\cdot|\mathbf{x}_i). \tag{2.1}$$

Here, $\gamma \in (0,1)$ is a discount factor that determines the exponential devaluation rate of delayed rewards. Note that the randomness in $D(\mathbf{x}_0)$ for any given state $\mathbf{x}_0$ is due both to the stochasticity of the sequence of states that follow $\mathbf{x}_0$, and to the randomness in the rewards $R(\mathbf{x}_0), R(\mathbf{x}_1), R(\mathbf{x}_2) \ldots$. We refer to this as the *intrinsic* randomness of the MDP. Using the stationarity of the MDP we may write

$$D(\mathbf{x}) = R(\mathbf{x}) + \gamma D(\mathbf{x}'), \text{ with } \mathbf{x}' \sim p^\mu(\cdot|\mathbf{x}). \tag{2.2}$$

The equality here marks an equality in the distributions of the two sides of the equation. Let us define the expectation operator $\mathbf{E}_\mu$ as the expectation over all possible trajectories and all possible rewards collected in them. This allows us to define the *value function* $V(\mathbf{x})$ as the result of applying this expectation operator to the discounted return $D(\mathbf{x})$. Let $\mathbf{E}_{\mathbf{x}'|\mathbf{x}} V(\mathbf{x}') = \int_{\mathcal{X}} d\mathbf{x}' p^\mu(\mathbf{x}'|\mathbf{x}) V(\mathbf{x}')$, and $\bar{r}(\mathbf{x}) = \int_{\mathbb{R}} dr q(r|\mathbf{x}) r$ be the expected reward at state $\mathbf{x}$. The value function satisfies the fixed-policy version of the Bellman Equation:

$$V(\mathbf{x}) = \bar{r}(\mathbf{x}) + \gamma \mathbf{E}_{\mathbf{x}'|\mathbf{x}} V(\mathbf{x}') \quad \forall \mathbf{x} \in \mathcal{X}. \tag{2.3}$$

2.1. The Value Model

The recursive definition of the discounted return (2.2) is the basis for our statistical generative model connecting values and rewards. Let us decompose the discounted return D into its mean V and a random, zero-mean residual ΔV,

$$D(\mathbf{x}) = V(\mathbf{x}) + \Delta V(\mathbf{x}), \tag{2.4}$$

where $V(\mathbf{x}) = \mathbf{E}_\mu D(\mathbf{x})$. In the classic frequentist approach $V(\cdot)$ is no longer random, since it is the true value function induced by the policy μ. Adopting the Bayesian methodology, we may still view the value $V(\cdot)$ as a random entity by assigning it additional randomness that is due to our subjective uncertainty regarding the MDP's model (p, q). We do not know what the true functions p and q are, which means that we are also uncertain about the true value function. We choose to model this additional *extrinsic* uncertainty by defining $V(\mathbf{x})$ as a random process indexed by the state variable $\mathbf{x}$. This decomposition is useful, since it separates the two sources of uncertainty inherent in the discounted return process D: For a known MDP model, V becomes deterministic and the randomness in D is fully attributed to the intrinsic randomness in the state-reward trajectory, modelled by ΔV. On the other hand, in a MDP in which both transitions and rewards are deterministic but otherwise unknown, ΔV becomes deterministic (i.e., identically zero), and the randomness in D is due solely to the extrinsic uncertainty, modelled by V. For a more thorough discussion of intrinsic and extrinsic uncertainties see Mannor et al. (2004).

Substituting Eq. (2.4) into Eq. (2.2) and rearranging we get

$$R(\mathbf{x}) = V(\mathbf{x}) - \gamma V(\mathbf{x}') + N(\mathbf{x},\mathbf{x}'),\ \mathbf{x}' \sim p^\mu(\cdot|\mathbf{x}), \tag{2.5}$$

where $N(\mathbf{x},\mathbf{x}') \stackrel{\text{def}}{=} \Delta V(\mathbf{x}) - \gamma \Delta V(\mathbf{x}')$. Suppose we are provided with a trajectory $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t$, sampled from the MDP under a policy μ, i.e., from $p_0(\mathbf{x}_0)\Pi_{i=1}^t p^\mu(\mathbf{x}_i|\mathbf{x}_{i-1})$, where p_0 is an arbitrary probability distribution for the first state. Let us write our model (2.5) with respect to these samples:

$$R(\mathbf{x}_i) = V(\mathbf{x}_i) - \gamma V(\mathbf{x}_{i+1}) + N(\mathbf{x}_i,\mathbf{x}_{i+1}),\ i = 0,\ldots,t-1. \tag{2.6}$$

Defining the finite-dimensional processes R_t, V_t, N_t and the $t \times (t+1)$ matrix $\mathbf{H}_t$

$$\begin{aligned} R_t &= (R(\mathbf{x}_0),\ldots,R(\mathbf{x}_t))^\top, \\ V_t &= (V(\mathbf{x}_0),\ldots,V(\mathbf{x}_t))^\top, \\ N_t &= (N(\mathbf{x}_0,\mathbf{x}_1),\ldots,N(\mathbf{x}_{t-1},\mathbf{x}_t))^\top, \\ \mathbf{H}_t &= \begin{bmatrix} 1 & -\gamma & 0 & \ldots & 0 \\ 0 & 1 & -\gamma & \ldots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \ldots & 1 & -\gamma \end{bmatrix}, \end{aligned} \tag{2.7}$$

we may write the equation set (2.6) more concisely as

$$R_{t-1} = \mathbf{H}_t V_t + N_t. \tag{2.8}$$

2.2. The prior

In order to specify a complete probabilistic generative model connecting values and rewards, we need

to define a prior distribution for the value process V and the distribution of the "noise" process N. We impose a Gaussian prior over value functions, i.e., $V \sim \mathcal{N}(0, k(\cdot,\cdot))$, meaning that V is a Gaussian Process (GP) for which, a priori, $\mathbf{E}(V(\mathbf{x})) = 0$ and $\mathbf{E}(V(\mathbf{x})V(\mathbf{x}')) = k(\mathbf{x},\mathbf{x}')$ for all $\mathbf{x},\mathbf{x}' \in \mathcal{X}$, where k is a positive-definite kernel function. Therefore, $V_t \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_t)$, where $\mathbf{0}$ is a vector of zeros and $[\mathbf{K}_t]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. Our choice of kernel function k should reflect our prior beliefs concerning the correlations between the values of states in the domain at hand.

2.3. The posterior

In order to maintain the analytical tractability of the posterior value distribution, we model the residuals $\Delta V_t = (\Delta V(\mathbf{x}_0), \ldots, \Delta V(\mathbf{x}_t))^\top$ as a Gaussian process. This means that the distribution of the vector ΔV_t is completely specified by its mean and covariance. Another assumption we make is that each of the residuals $\Delta V(\mathbf{x}_i)$ is generated independently of all the others. This means that, for any $i \neq j$, the random variables $\Delta V(\mathbf{x}_i)$ and $\Delta V(\mathbf{x}_j)$ correspond to two distinct experiments, in which two random trajectories starting from the states $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively, are generated independently of each other. We are now ready to proceed with the derivation of the distribution of the noise process N_t.

By definition (Eq. 2.4), $\mathbf{E}_\mu[\Delta V(\mathbf{x})] = 0$ for all $\mathbf{x}$, so we have $\mathbf{E}_\mu[N(\mathbf{x}_i, \mathbf{x}_{i+1})] = 0$. Turning to the covariance, we have $\mathbf{E}_\mu[N(\mathbf{x}_i, \mathbf{x}_{i+1})N(\mathbf{x}_j, \mathbf{x}_{j+1})] = \mathbf{E}_\mu[(\Delta V(\mathbf{x}_i) - \gamma\Delta V(\mathbf{x}_{i+1}))(\Delta V(\mathbf{x}_j) - \gamma\Delta V(\mathbf{x}_{j+1}))]$. According to our assumption regarding the independence of the residuals, for $i \neq j$, $\mathbf{E}_\mu\big[\Delta V(\mathbf{x}_i)\Delta V(\mathbf{x}_j)\big] = 0$. In contrast, $\mathbf{E}_\mu\big[\Delta V(\mathbf{x}_i)^2\big] = \mathbf{Var}_\mu\big[D(\mathbf{x}_i)\big]$ is generally larger than zero, unless both transitions and rewards are deterministic. Denoting $\sigma_i^2 = \mathbf{Var}\big[D(\mathbf{x}_i)\big]$, these observations may be summarized into the distribution of ΔV_t: $\Delta V_t \sim \mathcal{N}\big(\mathbf{0}, \mathrm{diag}(\boldsymbol{\sigma}_t)\big)$ where $\boldsymbol{\sigma}_t = [\sigma_0^2, \sigma_1^2, \ldots, \sigma_t^2]^\top$, and $\mathrm{diag}(\cdot)$ denotes a diagonal matrix whose diagonal entries are the components of the argument vector. In order to simplify the subsequent analysis let us assume that, for all $i \in \{1, \ldots, t\}$, $\sigma_i = \sigma$, and therefore $\mathrm{diag}(\boldsymbol{\sigma}_t) = \sigma^2\mathbf{I}$. Since $N_t = \mathbf{H}_t\Delta V_t$, we have $N_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)$ with,

$$\boldsymbol{\Sigma}_t = \sigma^2\mathbf{H}_t\mathbf{H}_t^\top = \sigma^2 \begin{bmatrix} 1+\gamma^2 & -\gamma & 0 & \ldots & 0 \\ -\gamma & 1+\gamma^2 & -\gamma & \ldots & 0 \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \ldots & -\gamma & 1+\gamma^2 \end{bmatrix}.$$

Since both the value prior and the noise are Gaussian, by the Gauss-Markov theorem (Scharf, 1991), so is the posterior distribution of the value conditioned on an observed sequence of rewards $\mathbf{r}_{t-1} = (r_0, \ldots, r_{t-1})^\top$. The posterior mean and variance of the value at some point $\mathbf{x}$ are given, respectively, by

$$\begin{aligned} \hat{v}_t(\mathbf{x}) &= \mathbf{k}_t(\mathbf{x})^\top\boldsymbol{\alpha}_t, \\ p_t(\mathbf{x}) &= k(\mathbf{x},\mathbf{x}) - \mathbf{k}_t(\mathbf{x})^\top\mathbf{C}_t\mathbf{k}_t(\mathbf{x}), \qquad (2.9) \\ \text{where } \boldsymbol{\alpha}_t &= \mathbf{H}_t^\top\left(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \boldsymbol{\Sigma}_t\right)^{-1}\mathbf{r}_{t-1}, \\ \mathbf{C}_t &= \mathbf{H}_t^\top\left(\mathbf{H}_t\mathbf{K}_t\mathbf{H}_t^\top + \boldsymbol{\Sigma}_t\right)^{-1}\mathbf{H}_t, \qquad (2.10) \end{aligned}$$

and $\mathbf{k}_t(\mathbf{x})$ is a vector of size t whose elements are $k(\mathbf{x}_i, \mathbf{x})$.

3. An On-Line Algorithm

Computing the parameters $\boldsymbol{\alpha}_t$ and $\mathbf{C}_t$ of the posterior moments (2.10) is computationally expensive for large samples, due to the need to store and invert a matrix of size $t \times t$. Even when this has been performed, computing the posterior moments for every new query point requires that we multiply two $t \times 1$ vectors for the mean, and compute a $t \times t$ quadratic form for the variance. These computational requirements are prohibitive if we are to compute value estimates on-line, as is usually required of RL algorithms. Engel et al. (2003) used an on-line kernel sparsification algorithm that is based on a view of the kernel as an inner-product in some high dimensional feature space to which raw state vectors are mapped. This sparsification method incrementally constructs a dictionary $\mathcal{D} = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_{|\mathcal{D}|}\}$ of representative states. Upon observing $\mathbf{x}_t$, the distance between the feature-space image of $\mathbf{x}_t$ and the span of the images of current dictionary members is computed. If the squared distance exceeds some positive threshold ν, $\mathbf{x}_t$ is added to the dictionary, otherwise, it is left out. Determining this squared distance, δ_t, involves solving a simple least-squares problem, whose solution is a $|\mathcal{D}| \times 1$ vector $\boldsymbol{a}_t$ of optimal approximation coefficients, satisfying

$$\boldsymbol{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \quad \delta_t = k_{tt} - \boldsymbol{a}_t^\top\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \quad (3.11)$$

where $\tilde{\mathbf{k}}_t(\mathbf{x}) = \big(k(\tilde{\mathbf{x}}_1, \mathbf{x}), \ldots, k(\tilde{\mathbf{x}}_{|\mathcal{D}_t|}, \mathbf{x})\big)^\top$ is a $|\mathcal{D}_t| \times 1$ vector, and $\tilde{\mathbf{K}}_t = \left[\tilde{\mathbf{k}}_t(\tilde{\mathbf{x}}_1), \ldots, \tilde{\mathbf{k}}_t(\tilde{\mathbf{x}}_{|\mathcal{D}_t|})\right]$ a square $|\mathcal{D}_t| \times |\mathcal{D}_t|$, symmetric, positive-definite matrix.

By construction, the dictionary has the property that the feature-space images of all states encountered during learning may be approximated to within a squared error ν by the images of the dictionary members. The threshold ν may tuned to control the sparsity of the

solution. Sparsification allows kernel expansions, such as those appearing in Eq. 2.10, to be approximated by kernel expansions involving only dictionary members, by using

$$\mathbf{k}_t(\mathbf{x}) \approx \mathbf{A}_t \tilde{\mathbf{k}}_t(\mathbf{x}), \quad \mathbf{K}_t \approx \mathbf{A}_t \tilde{\mathbf{K}}_t \mathbf{A}_t^\top. \tag{3.12}$$

The $t \times |\mathcal{D}_t|$ matrix $\mathbf{A}_t$ contains in its rows the approximation coefficients computed by the sparsification algorithm, i.e., $\mathbf{A}_t = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_t]^\top$, with padding zeros placed where necessary, see Engel et al. (2003).

The end result of the sparsification procedure is that the posterior value mean $\hat{v}_t$ and variance p_t may be compactly approximated as follows (compare to Eq. 2.9, 2.10)

$$\begin{aligned} \hat{v}_t(\mathbf{x}) &= \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\boldsymbol{\alpha}}_t, \\ p_t(\mathbf{x}) &= k(\mathbf{x},\mathbf{x}) - \tilde{\mathbf{k}}_t(\mathbf{x})^\top \tilde{\mathbf{C}}_t \tilde{\mathbf{k}}_t(\mathbf{x}), \end{aligned} \tag{3.13}$$

$$\text{where } \tilde{\boldsymbol{\alpha}}_t = \tilde{\mathbf{H}}_t^\top \left(\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\top + \boldsymbol{\Sigma}_t \right)^{-1} \mathbf{r}_{t-1}$$

$$\tilde{\mathbf{C}}_t = \tilde{\mathbf{H}}_t^\top \left(\tilde{\mathbf{H}}_t \tilde{\mathbf{K}}_t \tilde{\mathbf{H}}_t^\top + \boldsymbol{\Sigma}_t \right)^{-1} \tilde{\mathbf{H}}_t, \tag{3.14}$$

and $\tilde{\mathbf{H}}_t = \mathbf{H}_t \mathbf{A}_t$.

The parameters that the algorithm is required to store and update in order to evaluate the posterior mean and variance are now $\tilde{\boldsymbol{\alpha}}_t$ and $\tilde{\mathbf{C}}_t$, whose dimensions are $|\mathcal{D}_t| \times 1$ and $|\mathcal{D}_t| \times |\mathcal{D}_t|$, respectively. In many cases this results in significant computational savings, both in terms of memory and time, when compared with the exact non-sparse solution.

We omit the lengthy technical derivation of the algorithm as it appears in (Engel, 2005). The main idea is that if the matrix $\boldsymbol{\Sigma}_t$ is of a favorable shape, an efficient algorithm for recursive computation of the posterior can be derived. We note that the algorithm is expressed only in terms of kernel function evaluations, thus the input space $\mathcal{X}$ may be completely arbitrary.

4. Policy Improvement with GPSARSA

SARSA is a fairly straightforward extension of the TD algorithm (Sutton & Barto, 1998), in which state-action values are estimated, thus allowing policy improvement steps to be performed without requiring any additional knowledge on the MDP model. The idea is to use the stationary policy μ being followed in order to define a new, augmented process, the state space of which is $\mathcal{X}' = \mathcal{X} \times \mathcal{U}$, (i.e., the original state space augmented by the action space), maintaining the same reward model. This augmented process is Markovian with transition probabilities $p'(\mathbf{x}', \mathbf{u}' | \mathbf{x}, \mathbf{u}) = p^\mu(\mathbf{x}'|\mathbf{x})\mu(\mathbf{u}'|\mathbf{x}')$. SARSA is simply the TD algorithm applied to this new process. The same reasoning may be applied to derive a SARSA algorithm from the GP based TD algorithm. All we need is to define a covariance kernel function over state-action pairs, i.e., $k : (\mathcal{X} \times \mathcal{U}) \times (\mathcal{X} \times \mathcal{U}) \to \mathbb{R}$. Since states and actions are different entities it makes sense to decompose k into a state-kernel k_x and an action-kernel k_u: $k(\mathbf{x}, \mathbf{u}, \mathbf{x}', \mathbf{u}') = k_x(\mathbf{x}, \mathbf{x}')k_u(\mathbf{u}, \mathbf{u}')$. If both k_x and k_u are kernels we know that k is also a legitimate kernel (Schölkopf & Smola, 2002), and just as the state-kernel codes our prior beliefs concerning correlations between the values of different states, so should the action-kernel code our prior beliefs on value correlations between different actions.

All that remains now is to run the GP-based TD learning algorithm on the augmented state-reward sequence, using the new state-action kernel function. Action selection may be performed by ε-greedily choosing the highest ranking action, and slowly decreasing ε toward zero. However, we may run into difficulties trying to find the highest ranking action from a large or even infinite number of possible actions. This may be solved by fast iterative maximization method, such as the quasi-Newton method or conjugate gradients. Ideally, we should design the action kernel in such a way as to provide a closed-form expression for the greedy action (as in Engel et al., 2005).

5. Discussion

The nonparametric (kernel-based) Gaussian Process approach facilitates rich representations. Instead of focusing on Euclidean spaces, one can consider employing RL in essentially any space on which a kernel can be defined. Formally, the kernel is used to define a prior over the space of value functions. Intuitively, the kernel represents how "close" two states (or actions in the SARSA case) are which allows incorporating domain knowledge into the construction of the algorithm.

Finding "good" kernels is a challenge in kernel methods in general. However, there is no need for the kernel to be "optimal" for the algorithm to work well. All that is needed is for the kernel to be reasonable. In the kernel-methods community there is a significant body of work on how to create kernels for different kinds of objects. These include Fisher Kernels (Jaakkola & Haussler, 1998) for probabilistic models (the space $\mathcal{X}$ itself may be a space of probabilistic models), kernels for handwritten digits (DeCoste & Schölkopf, 2002), strings (Watkins, 1999), text documents (Joachims, 1998), genetic microarray data (Brown et al., 2000), sets of vectors (Kondor & Jebara, 2003) and trees (Collins & Duffy, 2001), to mention a few (see Shawe-

Taylor & Cristianini, 2004 for more extensive coverage).

The GP-based temporal difference approach itself has several distinct advantages. For example, it provides confidence bounds on the value function uncertainty. This can be potentially used for balancing exploration and exploitation. In Engel and Mannor (2005) we took advantage of the fact that the output parameters of GPTD algorithms are sufficient statistics for the posterior value process, summarizing the information contained in the observed trajectory, to extend the GP-based approach to a setup where multiple agents interact with the same environment. We finally mention that the policy improvement mechanism described above is SARSA based. It would be useful to devise a Q-Learning type algorithm for off-policy learning of the optimal policy; this is left for future research.

References

Brown, M., Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares, M., & Haussler, D. (2000). Knowledge-based analysis of microarray gene expression data by using suport vector machines. *Proceedings of the National Academy of Sciences* (pp. 262–267).

Collins, M., & Duffy, N. (2001). Convolution kernels for natural language. *Advances in Neural Information Processing Systems 14* (pp. 625–632).

DeCoste, D., & Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, *46*, 161–190.

Engel, Y. (2005). *Algorithms and Representations for Reinforcement Learning*. Doctoral dissertation, The Hebrew University of Jerusalem. `www.cs.ualberta.ca/∼yaki`.

Engel, Y., & Mannor, S. (2005). Collaborative temporal difference learning with Gaussian processes. Preprint.

Engel, Y., Mannor, S., & Meir, R. (2003). Bayes meets Bellman: The Gaussian process approach to temporal difference learning. *Proc. of the 20th International Conference on Machine Learning*.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. *Proc. of the 22nd International Conference on Machine Learning*.

Jaakkola, T., & Haussler, D. (1998). Exploiting generative models in discriminative classifiers. *Advances in Neural Information Processing Systems 11* (pp. 512–519).

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Proceedings of the Tenth European Conference on Machine Learning* (pp. 137–142).

Kondor, R. I., & Jebara, T. (2003). A kernel between sets of vectors. *Machine Learning, Proceedings of the Twentieth International Conference* (pp. 361–368).

Mannor, S., Simester, D., Sun, P., & Tsitsiklis, J. (2004). Bias and variance in value function estimation. *Proc. of the 21st International Conference on Machine Learning*.

Scharf, L. (1991). *Statistical signal processing*. Addison-Wesley.

Schölkopf, B., & Smola, A. (2002). *Learning with Kernels*. Cambridge, MA: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge, England: Cambridge University Press.

Sutton, R., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Watkins, C. (1999). *Dynamic alignment kernels* (Technical Report CSD-TR-98-11). UL Royal Holloway.

Bi-Memory Model for Guiding Exploration by Pre-existing Knowledge

Kary Främling KARY.FRAMLING@HUT.FI
Helsinki University of Technology, P.O. Box 5500, FI-02015 TKK, Finland.

Abstract

Reinforcement learning agents explore their environment in order to collect reward that allows them to learn what actions are good or bad in what situations. The exploration is performed using a policy that has to keep a balance between getting more information about the environment and exploiting what is already known about it. This paper presents a method for guiding exploration by pre-existing knowledge expressed by e.g. heuristic rules. A dual memory model is used where the value function is stored in long-term memory while the heuristic rules for guiding exploration act on the weights in a short-term memory. Experimental results from two "toy domains" illustrate that exploration is significantly improved when guidance can be provided by pre-existing knowledge.

1. Introduction

In supervised learning a "teacher" provides a set of training samples that have usually been pre-processed in a way that simplifies learning. Reinforcement learning (RL) differs from supervised learning mainly because the RL agent has to explore its environment by itself and collect training samples. The agent takes actions following a policy and observes received reward that it attempts to maximize. Except for simple tasks, this exploration can be long or even infeasible without any guidance. Different ways of providing such guidance has been studied by several researchers, e.g. Schaal (1997), Millán et al. (2002) and Driessens & Džeroski (2004).

This paper presents a method for exploring the environment using pre-existing knowledge expressed e.g. by heuristic rules. It uses a dual memory model where a so-called long-term memory is used for action-value learning and a so-called short-term memory is used for guiding action selection by heuristic rules. Experimental results with simple heuristic rules illustrate that it is possible to converge to a good policy with less exploration than with some well-known methods. This paper concentrates on episodic tasks, i.e. tasks with a pre-defined terminal state, even though the methods are not limited to such tasks.

After this introduction, the most relevant RL methods to the scope of this paper are described in Section 2. Section 3 presents methods to shorten initial exploration and how to combine them with methods presented in Section 2. Section 4 shows comparative results for three different tasks, followed by conclusions.

2. Reinforcement learning principles

Most existing RL methods try to learn a *value function* that allows them to predict the sum of future reward from any state s when following a given action selection policy π. Value functions are either *state-values* (value of a state) or *action-values* (value of an action in a given state). Action-values are denoted $Q(s,a)$, where a is an action. The currently most popular RL methods are so-called *temporal difference* (TD) methods (Sutton, 1988). *Q-learning* (Watkins, 1989) is a TD control algorithm that updates action-values according to

$$\Delta Q(s_t,a_t) = \beta\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right] e_{t+1}(s,a) \quad (1)$$

where $Q(s_t,a_t)$ is the value of action a in state s at time t, β is a learning rate, r_{t+1} is the immediate reward and γ is a discount factor. The max-operator signifies the greatest action-value in state s_{t+1}. $e_{t+1}(s,a)$ is an eligibility trace that allows rewards to be propagated back to preceding states and actions. A replacing eligibility trace (Singh & Sutton, 1996) is calculated according to

$$e_{t+1}(s) = \begin{cases} \gamma\lambda e_t(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases}, \quad (2)$$

where λ is a *trace decay* parameter that together with the γ determines how quickly rewards are propagated backwards. Another common method for action-value learning is SARSA (Rummery & Niranjan, 1994):

$$\Delta Q(s_t,a_t) = \beta\left[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right] e_{t+1}(s,a) \quad (3)$$

The most complete overviews available on methods for state-space exploration are (Thrun, 1992; Wiering, 1999).

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

Commonly used *undirected exploration* methods that do not use any task-specific information are *ε-greedy exploration* and *Boltzmann* action selection. ε-greedy exploration selects the greedy action with probability (1-ε) and an arbitrary action with probability ε using a uniform probability distribution. *Directed exploration* methods use task-specific knowledge for guiding exploration in such a way that the state space would be explored more efficiently. The action to be taken is selected by maximizing an evaluation function that combines action-values with *exploration bonus*es δ_n weighted by factors K_0 to K_k (Ratitch & Precup, 2003):

$$N(s,a) = K_0 Q(s,a) + K_1 \delta_1(s,a) + ... + K_k \delta_k(s,a) \quad (4)$$

Counter-based methods use an exploration bonus that directs exploration to less frequently visited states. *Recency-based exploration* prefers least recently visited states. Other directed exploration methods exist that use statistics on value function variance or other indicators. The same effect of preferring unexplored actions (and states) mainly in the beginning of exploration can be achieved by a technique called *optimistic initial values* that uses initial value function estimates that are bigger than the expected ones. A common implementation of this technique is to initialize action values to zero and give negative reward at every step (Thrun, 1992, p. 8). This means that unused actions have greater value estimates than used ones, so unused actions have a greater probability to be selected in the beginning of exploration.

3. SLAP reinforcement and the BIMM network

This section describes the use of a *long-term memory* (*LTM*) and *short-term memory* (*STM*)[1] model for combining action-value learning with heuristic rules that guide exploration (Främling, 2003). LTM is used for action-value learning while STM learning is used for guiding state-space exploration by the *SLAP* (Set Lower Action Priority) principle, described in sub-section 3.2. In sub-section 3.3 we study the effect of different learning parameters on the trade-off between rapid exploration and converging towards a "good" policy.

3.1 Bi-Memory Model (*BIMM*)

The bi-memory model uses a short-term memory for controlling exploration and a long-term memory for learning the value function. Both memories are here implemented as linear function approximators or Adalines (Widrow & Hoff, 1960), but any function approximator may be used for both STM and LTM. A linear function approximator calculates action values as the weighted sum of action neuron input values

$$a_j(s) = \sum_{i=1}^{N} s_i w_{i,j} \quad (5)$$

where s_i is the value of state variable i, $w_{i,j}$ is the weight of action neuron j for input i, a_j is the output value of action neuron j and N is the number of state variables. Weights are typically stored in a two-dimensional matrix of size $M \times N$, where M is the number of actions. This representation is identical to the lookup-table representation usually used in discrete RL tasks but gives the advantage of being able to handle continuous-valued state variables directly. Adalines can be trained using the Widrow-Hoff training rule

$$w_{i,j}^{new} = w_{i,j} + \alpha(a_j' - a_j)s_i \quad (6)$$

where a_j' is the "target" value used in supervised learning. α is a learning rate parameter that determines the step size of weight modifications. Widrow-Hoff learning is a gradient descent method that minimizes the root mean square error (RMSE) between a_j' and a_j samples. When using BIMM, Adaline outputs are calculated according to

$$a_j(s) = K_0 \sum_{i=1}^{N} ltw_{i,j} s_i + K_1 \sum_{i=1}^{N} stw_{i,j} s_i \quad (7)$$

where K_0 and K_1 are positive constants that control the balance between exploration and exploitation. STM is actually an exploration bonus whose influence on action selection is determined by the value of K_1 as in equation (4). $ltw_{i,j}$ is the LTM weight and $stw_{i,j}$ is the STM weight for action neuron j and input i. $a_j(s)$ is the estimated action-value $N(s,a)$ in equation (4). Both Q-learning and SARSA can be used to update LTM weights by replacing Q with ltw in equations (1) and (3).

3.2 Guiding exploration

Exploration bonuses affect action selection by increasing or decreasing the probability of an action being selected in a given state. If an action does not seem to be useful for exploration in some state according to some pre-existing knowledge, then make it less likely for that action to be used in that state. Similarly, if an action seems to be good in some state according to current rules, then make it more likely to be used in that state. In the tests performed in this paper only the first case is used, i.e. action probabilities are only decreased by the set lower action priority (SLAP) principle, where STM weights are updated using the Widrow-Hoff update rule with the target value

$$a_j'(s) = a_{min}(s) - margin \quad (8)$$

where $a_{min}(s)$ is the smallest $a_j(s)$ value in state s. The *margin* should have a "small" value (0.1 has been used in

[1] "Short-term memory" has been used in different contexts, e.g. for storing the eligibility trace; for memorizing parts of the state history in order to improve identification of "hidden states" (McCallum, 1995); and for storing contextual clues to be used in near-future states (Bakker 2002). These differ from the STM used here.

all tests reported in this paper), which ensures that an action that is repeatedly SLAPed will eventually have the lowest action value. Only STM weights are modified by the Widrow-Hoff rule, which becomes

$$stw_{i,j}^{new} = stw_{i,j} + \alpha(a_j' - a_j)s_i \quad (9)$$

The new activation value is then

$$\begin{aligned} &a_j^{new}(s) = K_0\sum_{i=1}^{N} ltw_{i,j}s_i + K_1\sum_{i=1}^{N} stw_{i,j}^{new}s_i = \\ &K_0\sum_{i=1}^{N} ltw_{i,j}s_i + K_1\sum_{i=1}^{N} s_i\left(stw_{i,j} + \alpha(a_j' - a_j)s_i\right) = \\ &K_0\sum_{i=1}^{N} ltw_{i,j}s_i + K_1\sum_{i=1}^{N} stw_{i,j}s_i + \alpha(a_j' - a_j)K_1\sum_{i=1}^{N} s_i^2 = \\ &a_j + \alpha(a_j' - a_j)K_1\sum_{i=1}^{N} s_i^2 \end{aligned} \quad (10)$$

where we can see that setting α to $1/(K_1\Sigma s_i^2)$ guarantees that a_j^{new} will become a_j' in state s after SLAPing action j. Replacing α with $\alpha/(K_1\Sigma s_i^2)$ in equation (9) gives a generalization for BIMM of the well-known Normalized Least Mean Squares (NLMS) method

$$stw_{i,j}^{new} = stw_{i,j} + \alpha(a_j' - a_j)s_i \Big/ K_1\sum_{i=1}^{N} s_i^2 \quad (11)$$

that reduces the error $(a_j' - a_j)$ exactly by the ratio given by α, which makes it easier to select a good value for α. For instance, if $\alpha = 1$ in equation (11) the SLAPed action will directly have the lowest $a_j(s)$ value for state s so it will not be used again until all other possible actions have been tested in the same state. This is especially useful in deterministic tasks. In stochastic tasks α should be inferior to one because even the optimal action may not always be successful, so immediately making its action-value the lowest would not be a good idea. As long as the value of $a_{min}(s)$ doesn't change, the value $a_j'(s)$ remains the same for all j. This is true until $a_j^{new}(s)$ becomes lower than the current $a_{min}(s)$ for some j. Therefore, action values slowly go towards minus infinity when using SLAP an infinite number of times[2].

A general algorithm for using SLAP in a learning task is given in Figure 1. Especially when using SARSA or other on-policy methods for action-value updates, STM weights should be updated before updating the action-values, otherwise the changes in STM weights might modify the action selection for the next state. It is also worth pointing out that the SLAP update rules do not include a time variable t as for Q-learning and SARSA, so SLAP can be used asynchronously with action-value updates. In fact, SLAP can be used for any state-action pair at any time independently of the current state and the current action selected.

```
Initialize parameters
REPEAT (for each episode)
  s ← initial state of episode
  REPEAT (for each step in episode)
    a ← action given by π for s
    Take action a, observe next state s'
    SLAP "undesired" actions
    Update action-value function in LTM
    s ← s'
```

Figure 1. General algorithm for using SLAP in typical RL task.

3.3 Increasing exploration

As shown by the experimental tasks in section 4, it can be easy to identify heuristic rules that make exploration faster. However, if these rules do not also give sufficient exploration of the state space, then it has to be provided by other means. STM weights are reinitialized before starting a new episode and can have a great impact on the way in which the state space is explored. Initializing STM weights to random values and using a "high" value for K_1 is one way of increasing exploration. In all tests reported here, STM weights have been initialized to random values in the interval [0,1) while LTM weights are initialized to zero. Therefore actions will initially be selected in a random order independently of the value of K_1 in equation (7) as long as LTM weights remain zero. When LTM values become non-zero due to action-value learning, the amount of randomness in action selection depends both on STM and LTM weights.

Undirected exploration methods (e.g. ε-greedy, Boltzmann) can also be used to increase exploration. Driessens and Džeroski (2004) alternated guided and unguided episodes for the same purpose, where hand-coded rules, human operators or other pre-existing knowledge provided guidance.

4. Experimental results

This section compares methods presented in the previous sections on two different tasks: 1) semi-MDP maze world and 2) mountain-car. Exploration methods compared are: **1) *Q*/*SARSA***: ε-greedy exploration; **2) *CTRB***: counter-based exploration **3) *OIV***: optimistic initial values and **4) *BIMM***. Q/SARSA, CTRB and BIMM use zero initial Q-values, $r = 1$ at terminal state and $r = 0$ for all other states. OIV used zero initial Q-values, $r = 0$ at goal and $r = -1$ for all other state transitions. Constant learning parameters are used in order to simplify the choice of parameter values and for reasons of comparability.

[2] Setting $a_j'(s) = a_{max}(s) + margin$ would increase weights in the same way that SLAP decreases them but this has not been useful for the experimental tasks in this paper.

4.1 Maze with transition delays

Semi-Markov Decision Processes (SMDP) may include continuous time (Bradtke and Duff, 1995). This signifies that the transition time from one state to another depends on a probability distribution F_{xy}. Here we introduce state transition time by the notion of "corridors" in a maze, i.e. states with only two actions that represent opposite directions. When in a corridor, the agent continues forward until it reaches a non-corridor state. The maze world is of size 20x20 (Figure 2), where 10 doors have been opened in addition to the initial unique solution. Both deterministic and stochastic state transitions are used. The stochastic state transition rates used are 0.2 and 0.5, which indicate the probability of another direction being taken than the intended one.

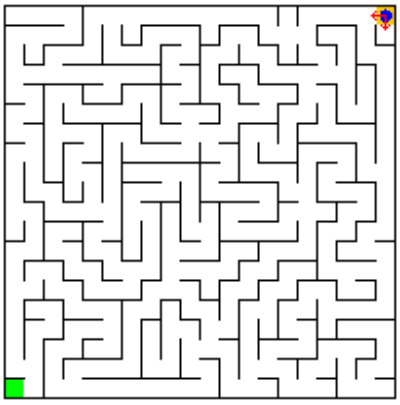


Figure 2. Maze with 10 supplementary "doors" opened in the walls in addition to the initial unique route. Agent in start position, goal position in lower left corner.

Q-learning without eligibility trace is used for action-value learning. Learning parameters are indicated in Table 1. The counter-based exploration bonus in equation (4) is implemented as

$$\delta_1(s,a) = \begin{cases} 1 - \dfrac{cnt(s,a) - cnt(s)_{\min}}{cnt(s)_{\max} - cnt(s)_{\min}} & \text{if } cnt(s)_{\max} - cnt(s)_{\min} > 0 \\ 0 & \text{if } cnt(s)_{\max} - cnt(s)_{\min} = 0 \end{cases} \quad (12)$$

where $cnt(s,a)$ is the counter for action a in state s and $cnt(s)_{min}$ and $cnt(s)_{max}$ are the smallest and greatest counter values for actions in state s. Counter values are reset after every episode. For BIMM agents, SLAP was used according to the following rules when entering a new state and before performing the next action: 1) SLAP the "inverse" action and 2) if the new state is already visited during the episode, SLAP action with the biggest value $a_j(s)$ in equation (7) for the new state. The rules are applied in the order indicated, so with $\alpha = 1$ the action taken at the previous visit becomes the last one in the action ranking given by equation (7).

All agents performed 250 episodes. Actions were selected greedily after 200 episodes in order to compare how well the value function was learned by all methods on an equal basis. This signifies that ε was set to zero for all agents as well as K_1 for BIMM and CTRB. Figure 3 shows that BIMM converges towards a good policy after much less exploration than Q- and CTRB-agents. The BIMM graph is close to the graph of the OIV agent but OIV converges very slowly. The greater the stochastic state transition rate, the slower the OIV agent converges. This is probably due to the cycles that occur with stochastic state transitions, which cause negative reward to be given even to the optimal action.

Table 1. Parameter values used in grid world tests. For BIMM $K_1 = 0.1$ in deterministic task and $K_1 = 10^{-6}$ in stochastic tasks. $\gamma = 0.95$ for all except OIV. Not indicated parameters are zero.

Agent	***Q***		***CTRB***		***OIV***		***BIMM***	
Grid world	β	ε	β	K_1	β	γ	α	β
Deterministic	1	0.1	1	0.1	1	1	1	1
Stoch. 0.2	0.1	0.1	0.5	0.01	0.5	0.95	0.2	0.5
Stoch. 0.5	0.1	0.1	0.5	0.001	0.5	0.95	0.1	0.5

Table 2 gives numeric comparisons for the performance of the four methods. BIMM agents achieve a better stable policy than the others in all tests as indicated by the third column. The total number of steps is also clearly lower than for the other agents. Since the training parameters often represent a compromise between how good the converged policy is and how much exploration is needed, the fact that BIMM has the best performance in both indicate its superiority in this task. Also, even though the advantage of BIMM in the beginning of exploration decreases with an increasing stochastic state transition rate (second column of Table 2), this is compensated by an improved "converged" policy compared with the other agents.

Table 2. Results for 20x20 maze with ten extra doors in the order Q/CTRB/OIV/BIMM. The third column indicates the average number of steps for the "stable" policy as average value of episodes 241-250.

Stoch. trans. rate	**Steps on first episode**	**Steps with converged policy**	**Total number of steps**
Deterministic	7450/4650/1760/**502**	49.0/**48.0**/**48.0**/**48.0**	134000/62800/27800/**21400**
0.2	7300/6200/2100/**1420**	64.4/69.1/69.2/**60.3**	145000/96000/57400/**41600**
0.5	9980/7840/**3900**/4170	116/196/132/**102**	152000/168000/105000/**103000**

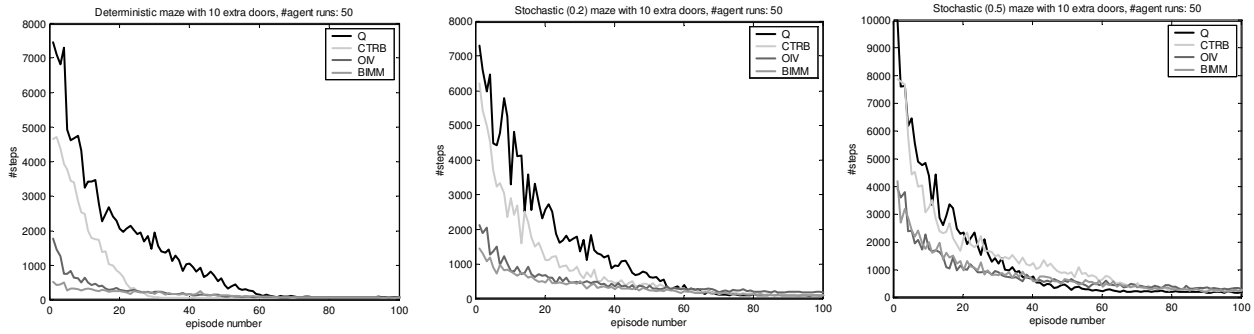


Figure 3. Maze results as average number of steps per episode. Grid size, stochastic transition rate and the number of agent runs used for calculating the average number of steps is indicated at the top of each graph.

4.2 Mountain-Car

The description of this task is similar to that in (Singh & Sutton, 1996) and (Randløv, 2000). The mountain-car task has two continuous state variables: the position x_t and the velocity v_t. At the beginning of each trial these are initialized randomly, uniformly from the range $x \in$ [-1.2,0.5] and $v \in$ [-0.07,0.07]. The altitude is $\sin(3x)$. The agent chooses from actions $a_t \in \{+1,0,-1\}$ that correspond to forward thrust, no thrust and reverse. The physics of the task are:

$$v_{t+1} = \text{bound}\left(v_t + 0.001a_t + g\cos(3x_t)\right)$$

and

$$x_{t+1} = \max\{x_t + v_{t+1}, -1.2\}$$

where $g = 0.0025$ is the force of gravity and the bound operation places the variable within its allowed range. If x_{t+1} is clipped by the max-operator, then v_{t+1} is reset to 0. The terminal state is any position with $x_{t+1} > 0.5$. The continuous state space is discretized by 8 non-overlapping intervals for each variable, which gives a total of 64 states. Episodes were limited to 1 000 000 steps.

As in most previous work on this task, the SARSA(λ) learning algorithm with replacing eligibility traces was used for action-value learning. Parameter values are: **SARSA**: learning rate $\beta = 0.1$, discount rate $\gamma = 0.9$, $\lambda = 0.95$, $\varepsilon = 0.1$; **OIV**: learning rate $\beta = 0.1$, discount rate $\gamma = 1.0$, $\lambda = 0.9$; and **BIMM**: learning rate $\beta = 0.1$, discount rate $\gamma = 0.9$, $\lambda = 0.95$, $K_1 = 0.1$, $\alpha = 1.0$. The counter-based method was not used in this task. With the continuous-valued state-variables it often takes several steps before changing state in the discretized state space, so an ordinary counter-based exploration changes the used action too rapidly to allow the agent to explore efficiently. This is also true for the heuristic rules used by BIMM in the maze task. It turns out that it is difficult to find good heuristics for the mountain-car task, as noticed by Randløv (2000) who tried to find a good reward shaping function.

The heuristic rules for using SLAP are 1) SLAP if sign of velocity is different from the sign of the action's thrust and 2) SLAP if velocity is positive and action is zero thrust. The second rule makes exploration slightly faster, but is not of much practical significance. These rules actually implement a controller that is at least nearly optimal for the task at hand, so one could ask what is the point of using a learning controller if we already have a good one? First of all, the initial controller is a static closed-loop controller while learning will give an adaptive open-loop controller that may be able to compensate for errors or calibration drift in real-world applications (see e.g. Främling (2004), for such an adaptive controller and Millán et al. (2002), where rules are used together with a learning controller). The initial controller may also be sub-optimal and incomplete, i.e. not cover the whole state space. Finally, the goal of this paper is to show that the BIMM and SLAP can be used for guiding exploration rather than evaluate the goodness of the heuristic rules.

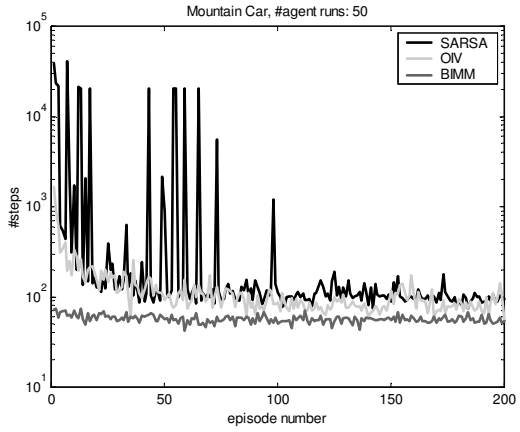


Figure 4. Average number of steps as a function of episode from 50 runs. The peaks are due to infinite episodes, limited to 1 000 000 steps. Note the logarithmical scale on the y-axis.

The results in Figure 4 are consistent with those in (Singh & Sutton, 1996) and (Randløv, 2000). SARSA uses an average of 38000 steps for the first episode, OIV uses 1700 steps and BIMM uses 73 steps. The simulations were run for 10000 episodes with greedy exploration from 9000 episodes onwards. The average numbers of steps for

the last 100 episodes (9901-10000) were 81.2 for SARSA, 78.6 for OIV and 55.6 for BIMM. The total numbers of steps for episodes 1 - 10000 are 1 140 000 for SARSA, 830 000 for OIV and 554 000 for BIMM so the BIMM agent clearly learned the action-value function better and with less exploration.

5. Conclusions

The results show that applying pre-existing knowledge through heuristic rules and the SLAP and BIMM mechanisms can make exploration more efficient both in deterministic and stochastic tasks, as well as in tasks involving continuous-valued state variables. For the heuristic rules used here, it is also apparent that benefits in exploration do not reduce the probability of learning a good value function. One big difference between BIMM and existing methods for improving exploration is that BIMM agents only use their own internal information about the task at hand. This makes them interesting compared with methods like reward shaping, which usually require some a priori knowledge about the environment, such as where the goal is located, the stochastic level of the environment or the number of sub-goals to reach.

Even though only "toy tasks" are used in this paper, it should be possible to generalize the results to many other RL tasks. This should be the case especially for tasks where state generalization is necessary due the number of states. Since SLAP and BIMM use standard ANN structures and learning rules, they are also applicable to tasks involving continuous-valued state variables and state classifiers. Such tasks are a subject of current and future research, where explosion of the state-space size due to state variable discretization is a problem.

References

Bakker, B. (2002). Reinforcement Learning with Long Short-Term Memory. In T. G. Dietterich, S. Becker, and Z. Ghahramani (eds.), *Advances in Neural Information Processing Systems 14*, MIT Press, Cambridge, MA. 1475-1482.

Bradtke, S.J., Duff, M.O. (1995). Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In G. Tesauro, D. Touretzky, T. Leen, (eds.), *Advances in Neural Information Processing Systems 7*, Morgan-Kaufmann. 393-400.

Driessens, K., Džeroski, S. (2004). Integrating Guidance into Relational Reinforcement Learning. *Machine Learning*, Vol. 57. 271–304.

Främling, K. (2003). *Guiding Initial State-space Exploration by Action Ranking and Episodic Memory*. Laboratory of Information Processing Science Series B, TKO-B 152/03, Helsinki University of Technology. http://www.cs.hut.fi/Publications/Reports/B152.pdf.

Främling, K. (2004). Scaled gradient descent learning rate - Reinforcement learning with light-seeking robot. *Proceedings of ICINCO'2004 conference*, 25-28 August 2004, Setúbal, Spain. 3-11.

McCallum, A. R. (1995). Instance-Based State Identification for Reinforcement Learning. In G. Tesauro, D. Touretzky, T. Leen (eds.) *Advances in Neural Information Processing Systems 7*, MIT Press, 1995. 377-384.

Millán, J.R., Posenato, D., Dedieu, E. (2002). Continuous-Action Q-Learning. *Machine Learning*, Vol. 49. 247-265.

Randløv, J. (2000). Shaping in Reinforcement Learning by Changing the Physics of the Problem. In *Proc. of ICML-2000 conference*. 767-774.

Ratitch, B., Precup, D. (2003). Using MDP Characteristics to Guide Exploration in Reinforcement Learning. In: *Lecture Notes in Computer Science*, Vol. 2837 (Proceedings of ECML-2003 Conference), Springer-Verlag, Heidelberg. 313-324.

Rummery, G. A., Niranjan, M. (1994). *On-Line Q-Learning Using Connectionist Systems*. Tech. Rep. CUED/F-INFENG/TR 166, Cambridge Univ. Engineering Department. 20 p.

Schaal, S. (1997). Learning from demonstration. In M. Mozer, M. Jordan, T. Petsche (eds), *Advances in Neural Information Processing Systems 9*, MIT Press. 1040-1046.

Singh, S.P., Sutton, R.S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, Vol. 22. 123-158.

Sutton, R.S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, Vol. 3. 9-44.

Thrun, S.B. (1992). The role of exploration in learning control. In DA White & DA Sofge, (eds.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, New York.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University.

Widrow, B., Hoff, M.E. (1960). Adaptive switching circuits. *1960 WESCON Convention record Part IV*, Institute of Radio Engineers, New York. 96-104.

Wiering, M. (1999). *Explorations in Efficient Reinforcement Learning*. Ph.D. thesis, University of Amsterdam. 218 p.

Why (PO)MDPs Lose for Spatial Tasks and What to Do About It

Terran Lane TERRAN@CS.UNM.EDU
Department of Computer Science, University of New Mexico, Albuquerque, NM 87131 USA

William D. Smart WDS@CSE.WUSTL.EDU
Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA

Abstract

In this deliberately inflammatory paper, we claim that everything you believe about (PO)MDPs is wrong. More specifically, we claim that (PO)MDPs are so general as to be nearly useless in many cases of practical interest and that we should specialize rather than generalize. We are mostly concerned with problems involving real, physical systems operating in a real, physical world (the same real, physical world that *we* live in). In particular, we are interested in spatial navigation, but we believe that this claim holds for a number of other key problem areas as well. Our abstraction efforts to date have focused on extending the reach of (PO)MDP models while maintaining their basic worldview. We claim that a profitable approach for the future is to cleave RL into a number of sub-disciplines, each studying important "special cases". By doing so, we will be able to take advantage of the properties of these cases in ways that our current (PO)MDP frameworks are unable to.

1. Provocative Claim

The (PO)MDP frameworks are fundamentally broken, not because they are insufficiently powerful representations, but because they are too *powerful. We submit that, rather than* generalizing *these models, we should be* specializing *them if we want to make progress on solving real problems in the real world.*

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Why (PO)MDPs are Broken

Reinforcement learning has lost track of its roots in the engineering of autonomous agents, and especially mobile robots. Ultimately, the family tree of our discipline roots back in the mathematicization of physics in the fifteenth and sixteenth centuries. For most of its history, this branch of science/engineering has been very closely coupled to the physical world. As a result, practitioners and theoreticians have almost universally depended on and exploited the properties of this environment. Often, that exploitation has been implicit, but the history of modern mathematics has, in some sense, been a quest to make explicit and formal the significant characteristics of these environments. As a result, those fields have developed extremely powerful and versatile abstractions of spatial environments. Ideas such as metrics, continuity, locality, topology, invariance, scale, etc. form core parts of disciplines from mathematics through physics and into a myriad of engineerings. But these ideas are precisely the ones that are missing from our formalisms in RL. We assert that the lack of these properties is actually a substantial barrier to achieving our goals.

In our efforts to formalize the notion of "learning control", we have striven to construct ever more general and, putatively, powerful models. By the mid-1990s we had (with a little bit of blatant "borrowing" from the Operations Research community) arrived at the (PO)MDP formalism (Puterman, 1994) and grounded our RL methods in it (Sutton & Barto, 1998; Kaelbling et al., 1996; Kaelbling et al., 1998). These models are mathematically elegant, have enabled precise descriptions and analysis of a wide array of RL algorithms, and are incredibly general. We argue, however, that their very generality is a hindrance in many practical cases. In their generality, these models have discarded the very qualities — metric, topology, scale, etc. — that have proven to be so valuable for many, many science and engineering disciplines. While we are just beginning to re-develop some of these concepts, situ-

ated within the (PO)MDP formalism, we are discovering that the (PO)MDP world-view actually *inhibits* our efforts to do so.

Consider the simple example of driving a car across a city. This is a task that is easy for humans[1], and for which our knowledge generalizes substantially. The knowledge you acquired when you first learned to drive (dynamics of the car in different conditions, written and unwritten traffic rules, navigational skills, etc.) is easily applied to a wide variety of driving tasks, including driving in situations that you have never previously encountered. The properties of this problem that you exploit to achieve this impressive performance are not well supported in the (PO)MDP framework. Specifically:

1. (PO)MDPs have no natural way to describe or represent the *metric* of an environment. Often, a global or local metric is simple to specify and rules out many possibilities (such as arbitrary teleportation to any state at any time point) that could be allowed under some legal (PO)MDP.

   A driver has a strong sense of both the "natural" metric (Euclidean or "as the crow flies" distance) and the "induced" metric (the graph of roads) that shape the driving environment. Thus, the driver can easily reason about locations that are near or far, discounting or neglecting the effects of things that are far away.

2. (PO)MDPs do not (natively) recognize the key differences between different *scales*. Physics has taught us that the world behaves very differently at subatomic, microscopic, macroscopic, and cosmological scales. Different abstractions, different mathematics, different models, and different predictions are necessary and appropriate for each. (PO)MDPs have no built-in notion of scale, and the rich literature on spatial and temporal abstraction (options (Sutton et al., 1999; Precup, 2000), HAMs (Parr & Russell, 1998), MAXQ (Dietterich, 2000), hierarchical POMDPs (Theocharous, 2002), etc.) have only offered partial solutions to date. These approaches all maintain the fundamental (PO)MDP worldview and, as a consequence, translate large, ugly, intractable (PO)MDPs into... Smaller, marginally less ugly, but still largely intractable (PO)MDPs. What is needed is a recognition of phase transitions between different scales.

   In the driving task, the world is stochastic at a local scale and the control dynamics of the vehicle dominate. At a routing scale, however, the world is essentially deterministic – drivers do route planning with static maps and completely neglect low-level effects like friction and vagaries of traffic patterns.

3. (PO)MDPs conflate properties of the *environment* with properties of the *agent*. Often, the environment itself has very well-defined properties that are not present in its (PO)MDP representation because of the stochasticity of agent dynamics. For example, spatial environments are metric, but the MDP representation can be non-metric. The agent's stochasticity, coupled with the need to average over all trajectories, can yield asymmetric "distances" – it is easily possible for the expected transition time from x to y to be different from the expectation of the reverse transition. By representing both agent and environment in a single framework, we have lost track of a very powerful property.

   Drivers draw a clear separation between properties of the environment – distances and the road graph, traffic rules and conditions, etc. – and properties of the agent – their own reaction time, dynamics of the vehicle, etc. This allows them to generalize large chunks of environment-dependent knowledge across different vehicles or chunks of control/dynamics knowledge across different environments (cities).

4. The (PO)MDP framework is tied to an *absolute coordinate frame*. Every atomic state has its own, potentially unique, transition and reward functions that must be learned, even though there may be many similar regions in the problem. For example, in the classic gridworld domain, there is one state per grid cell. However, many cells in the middle of the domain are "similar" in the sense that they have the same optimal local policy, in a relative coordinate frame. If we can recognize and exploit this similarity, we can make learning much more efficient and effective.

   In the physical world, one chunk of pavement is essentially indistinguishable from another chunk. All straight, two-lane roads are essentially the same, regardless of whether they occur in Chicago, St. Louis, or the middle of the Saskatchewan countryside. Traffic lights behave the same all over North America, as do yield signs. Humans exploit these regularities to translate policies among different driving experiences in different locations, but (PO)MDPs have no way to capture such generalizations.

[1] Except in Boston.

5. The (PO)MDP framework is strongly inclined toward *discrete state spaces*, where each state is only related to other states by its transition probabilities. Many problems have a natural ordering of states, but this is ignored by the general model. Recognition of this ordering means that we can effectively generalize from known to unknown states, and dramatically reduce the amount of experience we need to learn a good solution. Most commonly, we discretize a continuous state space or employ a function approximator (such as a neural network). In doing so, however, we often discard or neglect the metric properties of the world. Some techniques have embraced and exploited these properties (Perkins & Barto, 2001; Perkins & Barto, 2002), but we do not yet have a general approach to using the properties of continuous state spaces to improve our RL.

   Driving, is of course, a continuous problem and humans rely on that continuity to adapt control policies to situations that are "close" to what we have previously experienced. In most cases, it doesn't matter whether the vehicle is *precisely* centered within its lane or if it is a few centimeters left or right. The control policies are essentially the same in all of these cases. Further, while the general driving task is a highly nonlinear control problem, in sufficiently small regions, it *is* effectively linear and can be locally controlled with well-understood control theory.

We are hardly the first to have recognized many of these shortcomings. The rich body of work in different forms of RL abstraction is a sign of general recognition by the community of many of these flaws. Our contribution is the meta-claim: we should stop focusing on the completely general case and, instead, spend effort identifying *key properties* of important special cases and focus on building RL algorithms well-suited to those cases. In spatial navigation, for example, metric, scale, topology, etc., are natural properties to study and leverage. Other domains likely possess similar important domain constraints. In game playing, for example, the very basic principles of turn-taking and moving one piece per turn both dramatically constrain possible transition functions. In both cases, employing a highly general representation, like the (PO)MDP framework, forces our learning agent to spend immense amounts of valuable experience to *learn* properties that we already *know* a priori. The agent must prune out a large space of impossible models before it can focus on what we care about — optimizing performance within the space of relevant models.

We are not trying to suggest that the (PO)MDP framework is without merit or that there are no practical uses for it. Clearly, there are a number of very important practical successes of this approach. Many of our key insights about the theory of RL were also enabled by this view. And there certainly are, and will continue to be, systems that can *only* be effectively modeled via (PO)MDPs. But what we *are* suggesting is that the (PO)MDP framework is fundamentally the wrong level of abstraction for many tasks. In some sense, RL with (PO)MDPs is a bit like programming in assembly language — you can implement anything eventually, but it's excruciating and you would really rather program in Python.[2]

3. Current Results

We have begun some preliminary work to back up our inflammatory rhetoric. In this section, we give some rough sketches of this work, and then go on to outline a future research agenda.

3.1. Metric Environments and Envelopes

We have examined the properties of MDPs embedded in environments with metrics. Most "gridworld"-style domains that we have seen are (perhaps loosely) based on such environments, as are a number of more "realistic" tasks such as mountain car and tractor backup. We are interested in exploiting this metric to our advantage.

The basic intuition is that the metric of a space imposes a "speed limit" on the agent — the agent cannot transition to arbitrary points in the environment in a single step. Every step can reach, at most, some set of states "local" (with respect to the metric) to its current location. Thus, the transition function is quite sparse — most potential transitions have zero probability. This general observation has been made previously, but we have shown that this property can be used to derive a strong "envelope" bound (Dean et al., 1995). Specifically, we have shown that, when doing point-to-point navigation, an agent's trajectory remains within an elliptical envelope with high probability (Lane & Wilson, 2005). If the probability parameters are chosen correctly, the part of the state space beyond the envelope can be neglected for the purposes of planning because its impact on the agent's regret is insignificant. An example of such an envelope bound is shown in Figure 1 for an "open space" topology. The result can, however, be extended to more general

[2] Insert your own religiously held One True Programming Language here.

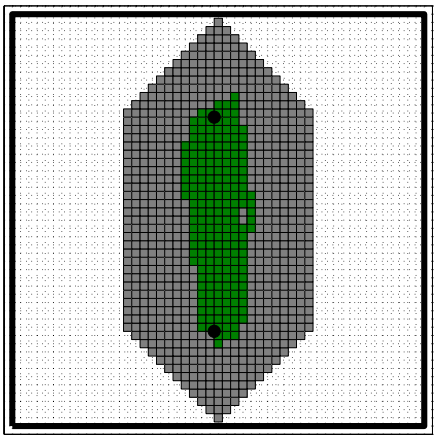


Figure 1. Illustration of the metric envelope bound for point-to-point navigation in an open-space gridworld environment. The outer, lighter gray region is the elliptical (with respect to Manhattan metric) envelope that contains 90% of the trajectory probability mass. The inner, darker region is the set of states occupied by an agent in a total of 10,000 steps of experience (319 trajectories from bottom to top). Note that the envelope bound is a bit loose, but succeeds in containing the agent's trajectories.

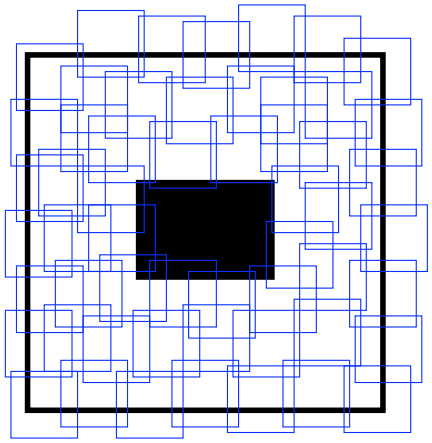


Figure 2. A simple navigation domain, covered with charts.

topologies. Interestingly, in a general metric topology, the bound remains an ellipse, but now with respect to that topology. E.g., the ellipse "wraps around the corner" to connect states separated by a wall segment.

The significance of this result is twofold. Most straightforwardly, an agent can neglect large parts of the state space when planning. More importantly, however, this result implies that *control* experience can be generalized across regions of the state space. If the agent learns a good policy for one bounded region of the state space, and it can find a second bounded region that is homeomorphic to the first (in the sense given by Ravindran and Barto (2002; 2003; 2004)), then the previously learned policy applies to the new region as well. When the environment is reasonably "regular" (having a large degree of repeated local structure), policy reuse could speed learning dramatically.

3.2. Learning and Planning with Manifold Representations

Our previous work has shown that a manifold-based value-function representation is effective for RL problems with continuous state spaces (Glaubius & Smart, 2004; Smart, 2004). Informally, a manifold representation models the domain of the value function using a set of overlapping local regions, called charts. Each chart has a local coordinate frame, is a (topological) disk, and has a (local) Euclidean distance metric. The collection of charts and their overlap regions is called a manifold.[3] We can embed partial value functions (and other models) on these charts, and combine them, using the theory of manifolds, to provide a global value function (or model). If we allocate these charts correctly, then they will model the topology of the problem domain. As an example, consider the simple navigation domain shown in Figure 2. Although the domain has a Euclidean coordinate frame, it is topologically cylindrical, because of the impassable area in the middle. The manifold formed by these charts is also topologically cylindrical.

Notice that the charts fall into thirteen equivalence classes (shown in Figure 3): those with nothing in them, those with an "inside" corner in them (4 classes), those with an "outside" corner (4 classes), and those with a straight wall.[4] This means that, with some suitable translations, we can learn models in these thirteen classes, and apply them to each instance of an actual chart. This can dramatically reduce the amount of experience that we need for learning. If we are willing to do some more work, we can reduce this to four equivalence classes (empty, "inside" corner, "outside" corner, and wall). However, this a more complex mapping between the world and the equivalence class, and might also necessitate transformations between actions.

We can also view this manifold as a graph, where the charts correspond to nodes, and the overlaps corre-

[3] This is not quite the whole story, but it's close enough. See our papers (Glaubius & Smart, 2004; Smart, 2004), or a general topology text (for example, Lefschetz (1949)) for the full details.

[4] We ignore areas outside of the world, so there are no walls with two sides.

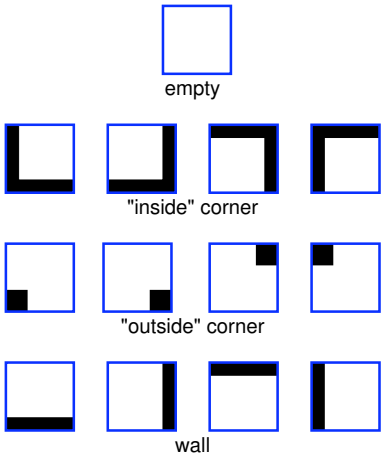


Figure 3. Thirteen chart equivalence classes. Taking advantage of rotational invariances as well could reduce this set to four.

spond to edges. This, combined with the local coordinate frames of the charts, allows us to define relative relationships between the charts. The graph representation also allows us to apply graph algorithms, allowing us to operate at the level of charts, which gives us a measure of abstraction.

On each of the charts, we can still operate at the level of raw (PO)MDPS, and rely on the manifold structure to blend the results between overlapping charts. We can treat the graph of charts as another (PO)MDP, where the actions correspond to local controllers in each of the charts. Once we have a chart-to-chart path, we can learn local controllers in each of the charts. Since each chart is locally Euclidean, we (claim that we) can use value-function approximation techniques and dense reward functions to accelerate learning within the chart. This will result in a set of partial controllers that will get us to the goal state. As we move towards the goal, we can also learn the transition function, T, for each chart. Once we reach the goal, we can calculate the actual (global) value function backwards through the charts, using our experiences, the "real" value function, and our learned transition model.

4. Research Agenda

The observations and (somewhat tongue in cheek) polemic of this paper lead us to the following research agenda: we intend to employ the tools of metric topology to develop efficient and powerful RL systems specialized for spatial navigation problems. Specifically, we are interested in the environments/tasks with the following properties:

metric For all x and y in the space, $d(x, y)$ is well defined. d need not be globally uniform — it can differ from region-to-region — but it must be possible to compare any pair of points.

topology Every point x in the space has a "local region" — a set of points within some d-ball around x. (This emerges as a consequence of the metric (Munkres, 1975), but is worth stating explicitly because it carries certain useful properties on its own.) The charts in Section 3.2 are examples of such regions.

regularity The environment has significant repeated substructure. E.g., there are many homeomorphic local regions (in the topological sense). Regularity improves experience reuse.

In addition, for many environments, it is useful to add:

locally linear Every topological region (e.g., chart) of the space is homeomorphic to a Euclidean disk. This allows a local coordinate frame for each region/chart and also supports function approximation via mapping to/from Euclidean space.

properties Every point is a member of one or more equivalence classes, denoted properties. Equivalence classes are defined in terms of the agent's dynamics at that point — all points for which the agent's transition function is homeomorphic are considered equivalent and assigned a property label. Properties can be common-sense quantities such as `mud` or `open-space`, or can be purely formal expressions of regularity in the environment. Properties allow us to easily capture and describe relevant facts about the agent's local dynamics and to generalize experience over the world.

These settings may be too restrictive/general, so part of our investigation will be analyzing the really useful set of criteria.

Given such environments, our first steps will be to combine the metric-based envelope bounds work (Section 3.1) with the chart/manifolds work (Section 3.2). The high-level idea is to first detect the presence of properties/equivalence classes in the environment by clustering transition experience. Then the environment can be segmented into charts at the equivalence class boundaries. The charts will be stitched together into a global manifold through a connectivity graph. RL will be used to develop short-range "local controllers" that can move from one chart to another. Then higher-level learning and planning can be done

on the manifold (graph) representation. The metric of the space, plus the envelope bounds, assure us that at least path planning can be done on this representation. Policy generalization at the manifold level will come from detecting repeated local structure (isomorphic subgraphs), where locality is again defined in terms of metric-based envelopes.

Ultimately (and ambitiously), we think that it will be possible to learn and plan at multiple levels of abstraction simultaneously. E.g., at the lowest, "atomic" level, we can learn about agent dynamics within individual charts. At the next level up we can learn the partition of the environment into charts and the relations among charts. Above that, we must recognize and generalize over collections of charts or subgraphs of the manifold. This problem is essentially a relational learning problem and is known to be quite challenging, but techniques for this are becoming available. At the very highest levels, it may be possible to work with fully deterministic models (such as graph planners, as proposed by Lane and Kaelbling (2002)) because the low-level uncertainty of the world will be abstracted away through long-run temporal averages.

References

Dean, T., Kaelbling, L. P., Kirman, J., & Nicholson, A. (1995). Planning under time constraints in stochastic domains. *Artificial Intelligence*, *76*.

Dietterich, T. G. (2000). An overview of MAXQ hierarchical reinforcement learning. In B. Y. Choueiry and T. Walsh (Eds.), *Proceedings of the symposium on abstraction, reformulation and approximation (sara 2000)*, Lecture Notes in Artificial Intelligence. New York: Springer Verlag.

Glaubius, R., & Smart, W. D. (2004). Manifold representations for value function approximation. *Proceedings of the the AAAI-04 Workshop on Learning and Planning in Markov Processes (AAAI Technical Report WS-04-08)* (pp. 13–18).

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–277.

Lane, T., & Kaelbling, L. P. (2002). Nearly deterministic abstractions of Markov decision processes. *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)* (pp. 260–266). Edmonton, Canada: AAAI Press.

Lane, T., & Wilson, A. (2005). Toward a topological theory of relational reinforcement learning for navigation tasks. *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*. Clearwater Beach, FL: AAAI Press. to appear.

Lefschetz, S. (1949). *Introduction to topology*. Princeton University Press.

Munkres, J. R. (1975). *Topology: A first course*. Prentice Hall.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems*. The MIT Press.

Perkins, T. J., & Barto, A. G. (2001). Lyapunov-constrained action sets for reinforcement learning. *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)* (pp. 409–416). Williams College, MA: Morgan Kaufmann.

Perkins, T. J., & Barto, A. G. (2002). Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, *3*, 803–832.

Precup, D. (2000). *Temporal abstraction in reinforcement learning*. Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: John Wiley & Sons.

Ravindran, B. (2004). *An algebraic approach to abstraction in reinforcement learning*. Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA.

Ravindran, B., & Barto, A. G. (2002). Model minimization in hierarchical reinforcement learning. *Proceedings of the 2002 Symposium on Abstraction, Reformulation, and Approximation (SARA-2002)*.

Ravindran, B., & Barto, A. G. (2003). Relativized options: Choosing the right transformation. *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 608–615). Washington, DC: AAAI Press.

Smart, W. D. (2004). Explicit manifold representations for value-functions in reinforcement learning. *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*. Paper number AI&M 25-2004.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Theocharous, G. (2002). *Hierarchical learning and planning in partially observable markov decision processes*. Doctoral dissertation, Michigan State University, East Lansing, MI.

A Hierarchical Approach to Efficient Reinforcement Learning

Michael L. Littman MLITTMAN@CS.RUTGERS.EDU
Carlos Diuk CDIUK@CS.RUTGERS.EDU
Alexander L. Strehl STREHL@CS.RUTGERS.EDU
Department of Computer Science, Rutgers University, Piscataway, NJ 08854-8019 USA

Abstract

Factored representations, model-based learning, and hierarchies are well-studied techniques for improving the learning efficiency of reinforcement-learning algorithms in large-scale state spaces. We bring these three ideas together in a new algorithm we call MaxQ-Rmax. Our algorithm solves two open problems from the reinforcement-learning literature. First, it shows how models can improve learning speed in the hierarchy-based MaxQ framework without disrupting opportunities for state abstraction. We illustrate the resulting performance gains in a set of example domains. Second, we show how hierarchies can augment existing factored exploration algorithms to achieve not only low sample complexity for learning, but provably efficient planning as well. We prove polynomial bounds on the computational effort needed by MaxQ-Rmax to attain near-optimal performance within the hierarchy with high probability.

1. Introduction

In the Markov decision process (MDP) formalization of the reinforcement-learning (RL) problem (Sutton and Barto 1998), a decision maker interacts with an MDP environment consisting of a finite state space S and action space A. Transitions are controlled by a Markov function where $P(s, a, s') = \Pr(s'|s, a)$ is the probability of reaching state $s' \in S$ after action $a \in A$ in state $s \in S$. The decision maker receives reward value $R(s, a)$ for action a in state s and attempts to maximize the expected cumulative reward.

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

In the current paper, we assume all MDPs consist of only negative rewards ($R(s, a) < 0$) except for a set of final states $F \subseteq S$ that end the process when reached. We further assume that there are *short* solutions—policies that achieve ϵ-optimal return with probability $1 - \delta$ within T steps for some relatively small T. This property holds for all discounted models, but also for more general models. It also implies that the optimal value function is unique.

A number of reinforcement-learning algorithms have been studied (Sutton and Barto 1998), some of which have been shown to find optimal policies under well-understood conditions. In analyzing algorithms, there are two principle sources of complexity to be considered. The first, *sample complexity*, defines the amount of real-world experience needed by an algorithm to achieve near optimal results. The second, *computational complexity*, specifies the amount of computational work required per experience. We seek algorithms with low sample and computational complexity—both bounded by polynomials in critical parameters of the environment.

DYNA (Sutton 1990) and prioritized sweeping (Moore and Atkeson 1993) showed how learning transition models can empirically decrease sample complexity at increased computational complexity. Recent work (Kearns and Singh 2002; Brafman and Tennenholtz 2002) has shown how model-based methods can provide formal polynomial-time bounds on both sample and computational complexity in MDPs.

Structure has long been recognized as important in computationally efficient sequential decision making (Boutilier *et al.* 1999). Dynamic Bayes Nets (DBNs) have emerged as a popular formalism for succinctly representing and solving large-scale MDPs (Koller and Parr 2000) and we adopt this factored-state framework in this paper.

In addition to structure in the state space, many researchers have studied algorithms that exploit hierar-

chical structure in the policy space (Kaelbling 1993; Hengst 2002). These methods have empirically provided relatively modest improvements in sample complexity over baseline RL approaches.

Kearns and Koller (1999) showed how model-based learning can be combined with factored states to provide polynomial sample-complexity bounds. An open problem from this work is achieving a polynomial computational-complexity bound. Dietterich (2000c) showed how factored states and hierarchy could be combined effectively, empirically improving sample complexity and reducing the number of parameters to be learned. Dietterich (2000b) also recognized the importance of combining models, factored states, and hierarchy, but found that the resulting learned hierarchical models no longer benefited from the factored representation.

The contribution of the current work is the combination of factored states, hierarchy and models, resulting in solutions to two important open problems. First, it shows how models can be combined with hierarchy without disrupting the benefits of factored states. The resulting algorithm exhibits greatly reduced sample complexity compared to model-free learning. Second, our combination of methods retains the polynomial sample complexity of existing combinations of models with factored states, with the hierarchy providing an approach to efficient planning in the learned models. Thus, we present the first factored-state reinforcement-learning algorithm with both polynomial sample *and* computational complexity. Section 4 illustrates our algorithm on a class of MDPs.

2. Factored-State MDPs

A factored-state MDP is one in which the state variables are factored into independently specified components. Let X be the set of state factors and, for all $x \in X$, $D(x)$ is the domain of values that factor x can take on. We write $v = \Phi_x(s)$ as the value of factor x in state s.

For a factored representation to provide a representational advantage over the standard tabular MDP representation, it is important that the transition probabilities and rewards support a structured representation. The assumption we adopt here, which generalizes the standard DBN representation, is that the probability that a factor x takes on a particular value v after a state transition in action a is a function of the *cluster* c of the state s.[1] We write $K(s,a)$ as the cluster for state s under action a and $\bar{P}^x(c,a,v)$ as the probability that a state from cluster c will transition to one that has factor x equal to v. Using this notation, for each action $a \in A$ and factor $x \in X$, transition probabilities are represented as $\Pr(s'|s,a) = \prod_{x \in X} \Pr(\Phi_x(s')|s,a) = \prod_{x \in X} \Pr(\Phi_x(s')|K(s,a),a) = \prod_{x \in X} \bar{P}^x(c,a,v)$ where $v = \Phi_x(s')$ and $c = K(s,a)$.

Similarly, we assume that reward functions are specified using the state clusters: $R(s,a) = \bar{R}(c,a)$ where $c = K(s,a)$.

2.1. Factored Rmax

Rmax is a reinforcement-learning algorithm introduced by Brafman and Tennenholtz (2002) and shown to have PAC sample complexity by Kakade (2003) (Brafman and Tennenholtz (2002) showed it was PAC in a slightly different setting). Factored Rmax is the direct generalization to factored MDPs (Guestrin *et al.* 2002). Factored Rmax is model based in that it maintains a model M' of the underlying factored MDP and, at each step, acts according to some optimal policy of its model. In this section, we'll describe the model used by Factored Rmax.

To motivate the model used by Factored Rmax, we'll describe at a high level the main intuition of the algorithm. Consider a fixed state factor x, action a, and cluster c ($c = K(s,a)$ for some state s). There exists an associated distribution $\bar{P}^x(c,a,\cdot)$ and reward $\bar{R}(c,a)$. The agent doesn't have access to these values, however, and they must be learned. The trick behind Factored Rmax is to use the agent's experience only when there is enough of it to ensure decent accuracy, with high probability.

Let κ be some user-defined constant that is given to Rmax as input at the beginning of a run. For each distribution $\bar{P}^x(c,a,\cdot)$, Rmax maintains a count $\kappa_a(x,v,c)$ of the number of times it has taken action a from a state s for which $K(s,a) = c$ and $v = \Phi_x(s)$. As long as $\kappa_a(x,v,c) < \kappa$, Rmax assumes that any transition from s under a causes state value x to become e, an additional value added to each domain $D(x)$. Once a state value becomes e, a transition under any action cannot change it from being equal to e. Additionally, the reward for any state that has a state variable with value e is equal to $R_{\max}$, the maximum possible reward (zero in our case). On the first timestep such that $\kappa_a(x,v,c) = \kappa$, Rmax updates its model to use the empirical distribution as an approximation of $\bar{P}^x(c,a,\cdot)$, and the empirical reward for $\bar{R}(c,a)$.

The crux of the argument proving that Factored Rmax has a polynomial sample complexity relies on the fol-

[1] In DBNs, the cluster is determined by by the joint assignment of the parents of X.

lowing techincal lemma, whose proof we omit here.

Lemma 1 *(Simulation Lemma) Let M be a factored MDP over n state variables and let $\hat{M}$ be an* **α-approximation** *of M, meaning that it is also a factored MDP with the same set of states, actions, DBNs, and reward function as M, and satisfies the condition that $|\hat{P}^x(c,a,\cdot) - \bar{P}^x(c,a,\cdot)| \leq \alpha$, for all actions a and clusters c, where $\hat{P}^x(c,a,\cdot)$ is the empirical distribution. There exists a constant κ such that if $\alpha = \kappa(\frac{\epsilon}{nT})$, then the value of the optimal policy π of $\hat{M}$ in the MDP M is ϵ-optimal.*

3. The MaxQ Value-Function Decomposition

One model of a task hierarchy in an MDP is to decompose the main task of maximizing reward en route to a terminal state into subtasks, each with its own terminal states and perhaps subtasks of its own.

Each task $1 \leq i \leq I$ in the hierarchy can be viewed as a self-contained MDP with final states F_i and action set A_i. Actions $j \in A_i$ can be either the primitive actions of the MDP or subtasks $j > i$. The root task $i = 1$ uses $F_i = F$ the final states of the actual MDP. For simplicity, each primitive action a is a task i with $A_i = \{a\}$ and $F_i = S$ (they terminate immediately after executing).

A *hierarchical policy* $\pi = \langle \pi_1, \ldots, \pi_I \rangle$ is a policy for each task i, $\pi_i : S \to A_i$. Policy π_i is considered locally optimal if it achieves maximum expected reward given subtask policies π_j for $j > i$. Similarly, we define a locally ϵ-optimal policy π_i as one that achieves within ϵ of the maximum expected reward given the fixed policies of its subtasks. If local optimality holds for all tasks, the corresponding hierarchical policy is called *recursively optimal*. We introduce the term *recursively ϵ-optimal* to mean that local ϵ-optimality holds at all levels of the hierarchy. Our Maxq-Rmax algorithm finds a recursively ϵ-optimal policy with polynomial sample and computational complexity.

3.1. Recursive Solution

The state (V) and state-action (Q) forms of the Bellman equations are well known (Sutton and Barto 1998). Dietterich (2000a) implicitly proposes an alternative—the completion-function form:

$$C(s,a) = \sum_{s'} P(s,a,s') \max_{a'}(R(s',a') + C(s',a')).$$

Given a representation for the reward function, the completion function can be used to recover $Q(s,a) = R(s,a) + C(s,a)$, $V(s) = \max_a Q(s,a)$, and $\pi(s) = \text{argmax}_a Q(s,a)$.

Given ϵ-approximate transition and reward functions $\hat{P}$ and $\hat{R}$ and the task hierarchy, we can compute a hierarchically ϵ-optimal policy by deifning a hierarchical completion function as follows. We consider the set of tasks i in reverse order from $i = I$ to $i = 1$. For each, we determine a completion function C^i. If i is the task for primitive action a, we define $C^i(s,a) = 0$.

For higher-level tasks i, we solve the MDP with actions A_i, states S, and final states F_i, where the transition function for action $j \in A_i$ is $P^j(s,s')$ and its reward function is $R^j(s)$. In other words, each subtask j is treated like an action that has a reward and a probabilistic transition to some state $s' \in F_j$. The MDP solution produces C^i. Specifically, for $s \in F_i$ we define $C^i(s,j) := 0$. Otherwise, we define the task completion function as follows:

$$C^i(s,j) = \sum_{s' \in F_j} P^j(s,s') \max_{j' \in A_i}(R^{j'}(s') + C^i(s',j'))$$

Computing $R^j(s)$ and $P^j(s,s')$ can be achieved by solving systems of linear equations: if $s \in F_j$, $P^j(s,s') = 1$ and $R^j(s) = 0$; otherwise,

$$\begin{aligned} P^j(s,s') &= \sum_{s_1} P^k(s,s_1) P^j(s_1,s') \quad \text{and} \\ R^j(s) &= R^k(s) + C^j(s,k) \end{aligned} \tag{1}$$

where $k = \pi^j(s)$.

In this construction, each task adopts an optimal policy given the subtasks, so $\pi = \langle \pi_1, \ldots, \pi_I \rangle$ is a recursively optimal policy for the MDP specified by $\hat{P}$ and $\hat{R}$.

It is well known that if a learning agent solves a model that is a close approximation to the true MDP, the agent can compute a near optimal policy from its model (Kearns and Singh 2002). This implies that a recursively optimal policy for an MDP with $\hat{P}$ and $\hat{R}$ close to P and R yields a recursively ϵ-optimal policy. To see this fact, note that at each step of the algorithm, the fixed policies of the descendents $j > i$ of the current task i imply an MDP for the current task. Consider any policy π for i given fixed policies for tasks $j > i$. This policy implies a Markov chain on the MDP M (where the states of M may be augmented with task information to achieve action restriction). Every transition in the Markov chain corresponds to a transition in M involving a primitive action. Thus, the result mentioned above applies to this setting, and a ϵ-optimal policy is found given the fixed policies of the current nodes descendant.

3.2. Abstraction

Dietterich (2000c) described how to combine state abstraction with a hierarchical task decomposition. For each task i, define $Z_i \subseteq S$ to be the set of abstract-state representatives, and define an *abstraction function* $\phi_i : S \to Z_i$ mapping states to their abstract representatives. We assume that abstract states are their own representatives: for all $z \in Z_i$, $\phi_i(z) = z$. An abstract policy and an abstract completion function differ only for states with different representatives.

Similar to Dietterich (2000c), we say an abstraction is *valid* if the abstract completion function does not incur any approximation under any abstract policy. Define the abstract transition function $\mathbb{P}_i^j : S \times Z_i \to \mathbb{R}$ as $\mathbb{P}_i^j(s, z') = \sum_{s' \in S \text{ s.t. } \phi_i(s')=z'} P^j(s, s')$. We adopt the following concrete assumptions, which imply a valid abstraction:

1. For all $s \in S$, $z' \in Z_i$ and $j \in A_i$, $\mathbb{P}_i^j(s, z') = \mathbb{P}_i^j(\phi_i(s), z')$. That is, the total probability of a transition to states in an abstract class is dependent only on the abstract class of the current state.
2. For all $s, s' \in S$, $j, k \in A_i$, if $P_i^j(s, s') > 0$, then $R^k(s') = R^k(\phi_i(s'))$. That is, for any reachable next state, the reward function depends only on the abstract class.

These assumptions imply a valid abstraction because the unique solution to the abstract completion function, defined by

$$\mathbb{C}^i(z, j) = \sum_{z' \in Z_i} \mathbb{P}_i^j(z, z') \max_{k \in A_i}(R^k(z') + \mathbb{C}^i(z', k)) \quad (2)$$

matches the completion function for all states $z \in Z_i$. This result follows from the uniqueness of the value function and algebraic substition using the assumptions above.

The abstract completion function can be used to compute task rewards and policies as follows. First, we extend the abstract completion function to all states in the obvious way: $\mathbb{C}^i(s, j) = \mathbb{C}^i(\phi_i(s), j)$. Next, we can define the reward function recursively by $R^i(s) = \hat{R}(s, a)$ if i is subtask for primitive action a, $R^i(s) = \max_{j \in A_i}(R^j(s) + \mathbb{C}^i(s, j))$ otherwise. Finally, these quantities can be used to define a policy via $\pi_i(s) = \text{argmax}_{j \in A_i}(R^j(s) + \mathbb{C}^i(s, j))$. Note that these computations can be performed in polynomial time in the size of the task hierarchy as long as results are cached (each task should only compute its reward function once for a state).

3.3. An Example Domain

The main MDP environment we used for testing is called **bitflip**. This domain has several qualities that make it suitable for our tests, including an exponential and scalable state space, an exponential set of Q values, linear set of completion values, and abstractions that can potentially reduce the size by an exponential factor. For the domain bitflip(n), meaning an instance of bitflip with n bits, the states are all binary n-bit numbers. There are n actions, with action flip(i) corresponding to "flip the ith bit". The actions don't always succeed. In particular, if action flip(i) is executed, the behavior depends on the bits to the left of i (j such that $j > i$). If any of these bits are set to 1, the action fails and all bits $j \geq i$ are set to 1. Otherwise, the ith bit is flipped with probably 0.75. With probability 0.25, nothing happens. The reward function is deterministic; whenever action i is executed, a reward of -2^i is received. There is one final state, the state where all bits are equal to zero. Note that $V(s) = -4/3\ s$ where s is the state interpreted as an integer in binary. Thus, there are 2^n distinct V and Q values.

We now describe the hierarchy for bitflip. For each i there is a subtask, clear(i), which terminates when the leftmost i bits are all zero. The subtasks available to clear(i) are flip(i) and clear($i - 1$). The subtask clear(n) is the root task. The abstraction function for flip(i) maps all states to four representative states. They correspond to the combinations of bit i being 0 or 1, and all bits to the left of i being 0 or at least having a 1. This abstraction is valid and results in an abstract completion function with fewer than $8n$ distinct values.

3.4. A Sampling-Based Hierarchical Planner

Section 3.1 describes how to find a recursively optimal policy given an MDP model and a task hierarchy. Via the completion function, it finds an optimal policy for each task by constructing a model of the lower-level tasks and treating them as primitive actions.

Equation 2 provides an alternative formulation of the completion function based on abstraction that can result in a considerably smaller representation at each task. Solving for the abstract completion function $\mathbb{C}$ requires knowledge of the abstract transition function $\mathbb{P}$ and the reward function R for each of the abstract state representatives. Unfortunately, the analog of Equation 1, which defines MDP models of subtasks, can no longer be formulated in the abstract setting since each task may have its own abstract state space.

Instead, we propose the following sampling-based scheme for estimating $\mathbb{P}$ and R sufficiently accurately to produce an ϵ-optimal policy for each task.

Once again, the algorithm proceeds by developing policies for each task i from $i = I$ down to $i = 1$. To compute the abstract completion function for task i, the algorithm estimates the abstract transitions and reward functions. For each subtask $j \in A_i$ and each abstract state representative $z \in Z_i$, the algorithm executes the policy π_j, recursively if necessary, from state z for up to T steps, recording the total reward r and final state s' reached. If no final state is reached in T steps, r is set to zero. This process is repeated m times, for a value of m to be specified below. After the sampling process is complete (at most mT steps), the algorithm computes $\hat{\mathbb{P}}_i^j(z, z')$ as the fraction of the m runs in which a final state s' was reached such that $\phi_i(s') = z'$. Similarly, $\hat{R}^j(z)$ is the average of the r values over the m runs.

Using these estimates for $\mathbb{P}$ and R, Equation 2 can be used to compute the abstract completion function, thus concluding the policy determination for task i.

There are several key insights used to prove the efficiency of the algorithm described above. We provide only a sketch of the argument needed. We make the simplifying assumption that rewards are deterministic (relaxing this assumption is possible) and bounded above -1. First, note that if the agent has a model that is an (ϵ/T)-approximation of the true MDP, then an optimal policy based of its model will be ϵ-optimal (Kakade 2003) in the true MDP. Second, we note that an individual sample trajectory of the type above can be viewed as a random draw from some distribution (over next representative states). Thus, an application of any bound on the L_1 distance between the empirical and true distribution can be applied to choose a valid m. For example, Kakade (2003) used one to prove that Rmax has polynomial sample complexity.

4. Preliminary Empirical Results

We tested MaxQ and Factored Rmax in the bitflip domain. We've also implemented MaxQ-Rmax, but the results are too preliminary to report at this time. The results show that, as expected, Rmax has poor computational complexity and MaxQ has poor sample complexity. We hope to report positive empirical results for MaxQ-Rmax by the workshop.

Our testing methodology was to evaluate the agent's learned policy after each episode on a number of example states of the environment. The evaluation states were picked to follow a standard structure based on

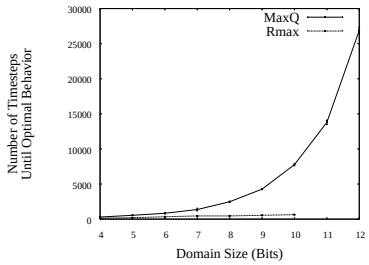


Figure 1. Number of steps per problem size until algorithm reaches optimal policy for the given start states.

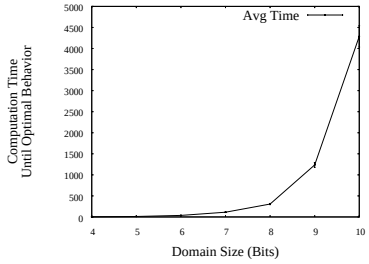


Figure 2. Average running time per step in Factored-Rmax.

the number of bits n: all 1s (11..11), single 1 in the center (0..010..0), first half 1s (1..10..0), second half 1s (0..01..1), by quarters (1.10.01.10.0) and alternating 1s (101010...). Once an optimal policy is found, the trial stops and we record the length (number of action choices) and computer time for the trial. Each experiment is repeated for twenty trials and all the results are averaged. We tested each algorithm on bitflip(n), where we allowed n to increase until one of two termination conditions were satisfied: when a value of n is reached for which the algorithm took longer than thirty minutes on a single trial, or when the number of action choices reached thirty thousand.

Each algorithm has a number of parameters. For Factored Rmax, the main parameter is the Rmax constant C. For MaxQ, one must specify the value of ϵ for exploration and its decay rate. We did a broad search of these parameters for each domain and chose the setting that gave the best performance for each algorithm and problem size.

As we can see from the Figure 1, the sample complexity of Factored Rmax grows linearly in the number of factors of the domain, yet the computational cost grows exponentially (Figure 2). As expected, the inverse holds for MaxQ (Figures 1 and 3).

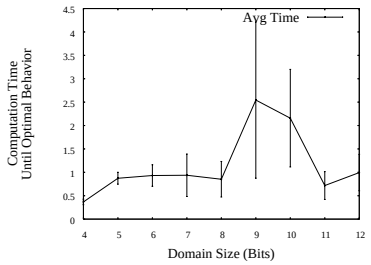


Figure 3. Average running time per step in MaxQ.

5. Conclusion

We have designed a novel algorithm that combines factored representations, model-based learning, and hierarchies to provide a formal guarantee on learning time in many large reinforcement-learning domains. Empirical results indicate that, for a simple extensible class of MDPs, existing algorithm fail to provide either efficient sample complexity or efficient computational complexity. Ongoing empirical work will demonstrate how the theoretical assurances of our hybrid algorithm translate to practical advantages.

References

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[Brafman and Tennenholtz, 2002] Ronen I. Brafman and Moshe Tennenholtz. R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[Dietterich, 2000a] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[Dietterich, 2000b] Thomas G. Dietterich. An overview of MAXQ hierarchical reinforcement learning. In *Proceedings of the Symposium on Abstraction, Reformulation and Approximation SARA 2000*, pages 26–44, 2000.

[Dietterich, 2000c] Thomas G. Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems 12*, pages 994–1000, 2000.

[Guestrin *et al.*, 2002] Carlos Guestrin, Relu Patrascu, and Dale Schuurmans. Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. In *Proceedings of the International Conference on Machine Learning*, pages 235–242, 2002.

[Hengst, 2002] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Maching Learning: Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250. Morgan Kaufmann, 2002.

[Kaelbling, 1993] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA, 1993. Morgan Kaufmann.

[Kakade, 2003] Sham M. Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

[Kearns and Koller, 1999] Michael J. Kearns and Daphne Koller. Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 740–747, 1999.

[Kearns and Singh, 2002] Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2–3):209–232, 2002.

[Koller and Parr, 2000] Daphne Koller and Ronald Parr. Policy iteration for factored MDPs. In *Uncertainty in Artificial Intelligence, Proceedings of the Sixteenth Conference (UAI 2000)*, pages 326–334, 2000.

[Moore and Atkeson, 1993] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, 1998.

[Sutton, 1990] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, Austin, TX, 1990. Morgan Kaufmann.

Hierarchical Reinforcement Learning Using Graphical Models

Victoria Manfredi VMANFRED@CS.UMASS.EDU
Sridhar Mahadevan MAHADEVA@CS.UMASS.EDU
Computer Science Dept, 140 Governor's Drive, University of Massachusetts, Amherst, MA 01003-9264 USA

Abstract

The graphical models paradigm provides a general framework for automatically learning hierarchical models using Expectation-Maximization, enabling both abstract states and abstract policies to be learned. In this paper we describe a two-phased method for incorporating policies learned with a graphical model to bias the behaviour of an SMDP Q-learning agent. In the first reward-free phase, a graphical model is trained from sample trajectories; in the second phase, policies are extracted from the graphical model and improved by incorporating reward information. We present results from a simulated grid world Taxi task showing that the SMDP Q-learning agent using the learned policies quickly does as well as an SMDP Q-learning agent using hand-coded policies.

1. Introduction

Abstraction is essential to scaling reinforcement learning (RL) (Barto & Mahadevan, 2003; Dietterich, 2000; Parr & Russell, 1998; Sutton et al., 1999). Temporal abstraction permits structured initial exploration by RL agents, allowing reuse of learned activities, and simplifying human interpretation of the learned policy. Spatial abstraction decreases the number of states that need to be experienced, reducing the amount of memory needed, and capturing regularities in the policy structure. Given predefined state and policy abstractions, for instance a task hierarchy, existing methods (Dietterich, 2000; Parr & Russell, 1998; Sutton et al., 1999) can be used to learn the corresponding hierarchical policy. One of the most difficult problems in hierarchical reinforcement learning, however, is how to automatically learn the abstractions. For instance, suppose a MAXQ hierarchy is desired: What should the tasks be? How should the termination conditions be defined?

The graphical models framework provides a powerful approach to automatically learning abstractions for hierarchical RL. For example, the abstract hidden Markov model (AHMM) proposed by Bui *et al.* (2002) is a hierarchical graphical model that encodes abstract policies; these policies are derived from the options framework (Sutton et al., 1999) and have associated initiation and termination states. Alternatively, the hierarchical hidden Markov model (HHMM) (Fine et al., 1998) encodes abstract states.

In this work, we describe the use of graphical models to automate hierarchical reinforcement learning using imitation. The method we propose takes advantage of a mentor who provides examples of optimal behaviour in the form of state transitions and primitive actions. We believe humans exploit similar guidance when learning complex skills, such as driving.

Previous approaches to automatic abstraction in RL can be divided into two groups: methods that identify subgoals, i.e., states or clusters of states, and then learn policies to those subgoals (McGovern & Barto, 2001; Şimşek & Barto, 2004; Mannor et al., 2004) and methods that explicitly build a policy hierarchy (Hengst, 2002). Key differences between our method and previous work are that first, by modifying the structure of the graphical model, different abstractions can be learned; second, we learn sub-goals (terminations) and policies simultaneously, rather than separately; finally, our method provides a mechanism for coping with partially observable state through the use of hidden variables.

Previous work on imitation in the context of RL has focused on learning a flat policy model. In Price and Boutilier (2003) a mentor's state transitions are used to help the learner converge more quickly to a good policy. In Abbeel and Ng (2004), the observer's reward function is unknown; instead, a mentor's state

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

transitions and feature expectations are used to identify a policy with similar feature expectations, where features are a mapping over states and feature expectations partially encode the value of the policy. In comparison, graphical models provide a mechanism for learning by imitation where the mentor learns not to just imitate the teacher but learns the task structure implicit in the higher level subgoals in the mentor's policy. This inference involves computing a distribution over the mentor's higher-level policies from its state transitions and actions with higher level policy variables treated as hidden variables. While not studied here, rewards could additionally be incorporated into the graphical model, as in (Samejima et al., 2004).

In the rest of this paper, we define the graphical model that we use and investigate its effectiveness in automating hierarchical RL.

2. Dynamic Abstraction Networks

Previous work in graphical models has largely focused on studying temporal abstraction or state abstraction in isolation. Intuitively, abstract policies are intricately tied to abstract states. For instance, New York City is both a temporal and a spatial abstraction: its geographical location permits you to both execute such abstract policies as visit the Metropolitan Museum of Art or attend a Broadway play, and to define such spatial abstractions as the five boroughs of New York City or the state of New York.

In other work (Manfredi & Mahadevan, 2005) we have proposed a new type of hierarchical graphical model, *dynamic abstraction networks* (DANs), that combines state and temporal abstraction. Jointly encoding state and temporal abstraction permits abstract states and policies to be learned simultaneously, unlike in the AHMM or HHMM alone. We showed in (Manfredi & Mahadevan, 2005) that empirically DANs seem to learn better policy abstractions than do AHMMs.

Figure 1 shows the dynamic Bayesian network (DBN) representation of a 2-level DAN. Informally, we can think of DANs as a merging of a state hierarchy, represented by the HHMM, and a policy hierarchy represented by a modified version of the AHMM which we refer to as an mAHMM. Technically we use the dynamic Bayesian network representations of the HHMM and AHMM defined in (Murphy & Paskin, 2001) and (Bui et al., 2002) respectively. We merge the mAHMM and HHMM by adding edges from state nodes at time t on level i to policy and policy termination nodes at time t on level i and from policy nodes at time t on level i to state nodes at time $t+1$ on

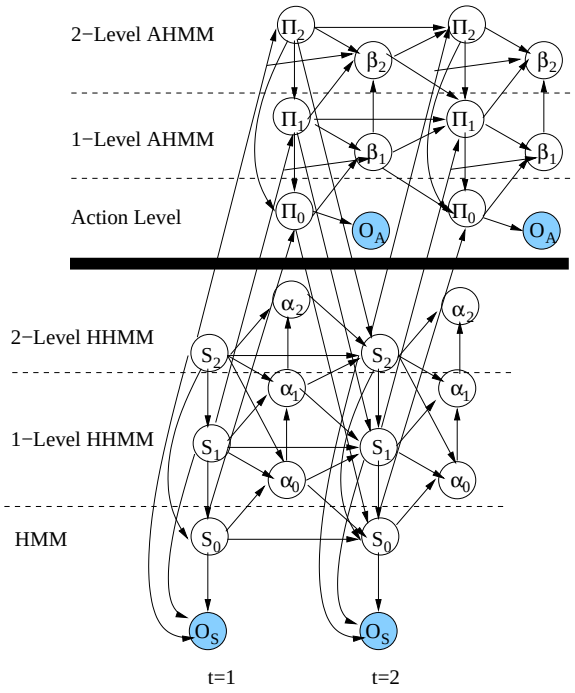


Figure 1. DBN representation of a dynamic abstraction network. We emphasize that this is just one realization of a dynamic abstraction network, and other configurations are possible.

level i. One of the key ideas for developing this model is that abstract states are useful for learning abstract policies. Consequently, abstract states are fed into all policy levels including the actions. We formally define a K-level DAN as comprised of an intertwined state hierarchy and policy hierarchy defined as follows.

Policy Hierarchy. A policy hierarchy H_π with K levels is given by the ordered tuple $H_\pi = (O_A, \Pi_0, \Pi_1, \ldots, \Pi_K)$.

- O_A is the set of action observations. O_A is only necessary if actions are continuous or partially observable.
- Π_0 is the set of primitive (one-step) actions.
- Π_i is the set of abstract policies at level $1 \leq i \leq K$, defined in more detail below.

State Hierarchy. A state hierarchy H_s with K levels is given by the ordered tuple $H_s = (O_S, S_0, S_1, \ldots, S_K)$.

- O_S is the set of state observations.
- S_j is the set of abstract states at levels $0 \leq j \leq K$, defined in more detail below.

Abstract Policies. At level i, each abstract policy $\pi_i \in \Pi_i$ is given by the tuple $\pi_i =< S_{\pi_i}, B_{\pi_i}, \beta_{\pi_i}, \sigma_{\pi_i} >$.

- *Selection set.* $S_{\pi_i} \subset S_i$ is the set of states in which π_i can be executed.
- *Termination probability.* $B_{\pi_i} \subset S_i$ is the set of states in which π_i can terminate. $\beta_{\pi_i} : B_{\pi_i} \rightarrow (0, 1]$ is the probability that policy π_i terminates in state B_{π_i}. Note that for all termination states that are not also selection states, policy π_i terminates with probability one.
- *Selection probability.* $\sigma_{\pi_i} : \Pi_{i+1:K} \times S_{\pi_i} \rightarrow [0, 1]$ is the probability with which a policy $\pi_i \in \Pi_i$ can be initiated when executing abstract policies $\pi_{i+1} \in \Pi_{i+1}, \ldots, \pi_K \in \Pi_K$ in state $s_{\pi_i} \in S_{\pi_i}$.

Abstract States. At level j, each abstract state $s_j \in S_j$ is given by the tuple $s_j =< \Pi_{s_j}, \alpha_{s_j}, \sigma_{s_j} >$. Note that an abstract state may be part of more than one higher level state; for instance, states at level j may be letters while states at level $j + 1$ are words.

- *Entry set.* $\Pi_{s_i} \subset \Pi_i$ is the set of policies which enter state s_i.
- *Exit probability.* $\alpha_{s_j} : S_j \rightarrow (0, 1]$ is the probability that the agent exits state $s_j \in S_j$ when in states $s_{j+1} \in S_{j+1}, \ldots, s_K \in S_K$.
- *Transition probability.* $\sigma_{s_j} : S_{j+1:K} \times \Pi_j \times S_j \rightarrow [0, 1]$ is the probability that the agent transitions to state $s_j^{t+1} \in S_j$ from state $s_j^t \in S_j$ when in parent states $s_{j+1}^t \in S_{j+1}, \ldots, s_K^t \in S_K$ and executing policy $\pi_j^t \in \Pi_j$.

For the policy hierarchy, Π nodes encode the *selection sets* S_π and the *selection probabilities* σ_π, while β nodes encode the *termination probabilities* β_π. For the state hierarchy, S nodes encode the *entry sets* Π_s and the *transition probabilities* σ_s, while α nodes encode the *exit probabilities* α_s. At level i, choosing policy $\pi_i \in \Pi_i$ depends in part on state $s_i \in S_i$ but executing $\pi_i \in \Pi_i$, by choosing policy $\pi_{i-1} \in \Pi_{i-1}$, depends in part on state $s_{i-1} \in S_{i-1}$. That is, choosing a policy at level i depends on the state at level i, but executing a policy at level i depends on the state at level $i - 1$.

3. Hierarchical Reinforcement Learning

In this section, we show how DANs can be applied to the problem of automating hiearchical RL.

3.1. Approach

There are two phases to our approach. In the first phase, the DAN is constructed and trained. Like MAXQ (Dietterich, 2000), our approach requires that the number of levels and the number of policies and states at each level be specified. However, unlike MAXQ, the dependencies between policies and states at different levels are unknown. Instead, all policies (states) at one level are connected to all policies (states) in the adjacent levels to permit any possible set of dependencies to be learned. Sequences of state-action pairs obtained from a mentor are then used to train the model with Expectation-Maximization (EM) (Dempster et al., 1977). Unlike other approaches for learning hierarchies, by reducing the problem to parameter estimation, all levels of the state and policy hierarchies are learned simultaneously through joint inference on the model. Learning in the first phase is on-policy; consequently, the quality of the sample trajectories used to train the model will affect the quality of the policies learned.

In the second phase, the learned policies are obtained from the DAN and improved using RL. Once the DAN has been trained, the policy hierarchy is extracted. The options framework (Sutton et al., 1999) fits most naturally with the DAN policy hierarchy. Changing the policy hierarchy would permit other types of policy hierarchies, such as HAMs (Parr & Russell, 1998) or MAXQ task graphs (Dietterich, 2000), to be used. An option consists of (1) a set of states in which it can be initiated, (2) a set of states in which it terminates, (3) a probability distribution over termination states for each option, and (4) a probability distribution over actions (or lower-level options) for each state. Define a 1-level option to be a policy over options. Then an i-level option is encoded within an DAN: Π nodes encode (1), (2), and (4), while β nodes encode (3). Various methods could be used, but we use reward to improve the extracted policies by using semi-Markov decision process (SMDP) Q-learning (Sutton et al., 1999) to estimate the value function. We discuss this further in the Experiments section.

We note that like DANs, MAXQ similarly encodes state abstractions with associated policy abstractions (task decompositions). However, unlike DANs, these abstractions are hand-specified rather than learned.

3.2. Experiments

The *Taxi* domain (Dietterich, 2000) is used to illustrate the proposed approach. The *Taxi* domain (Dietterich, 2000) consists of a five-by-five grid with four taxi terminals, R, G, Y, and B, see Figure 2. The goal

R 1	2	3	4	G 5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
Y 21	22	23	B 24	25

Figure 2. Taxi grid.

of the agent is to pick up a passenger from one terminal, and deliver her to another one (possibly the same one). There are six actions: north (N), south (S), east (E), west (W), pick up passenger (PU), and put down passenger (PD). 80% of the time N, S, E, and W work correctly; for 10% of the time the agent goes right and 10% of the time the agent goes left. The agent's state consists of the taxi location (TL), the passenger location (PL), and the passenger destination (PD). Note that $PL = 1$ when the passenger has been picked up and $PD = 1$ when the passenger has been delivered.

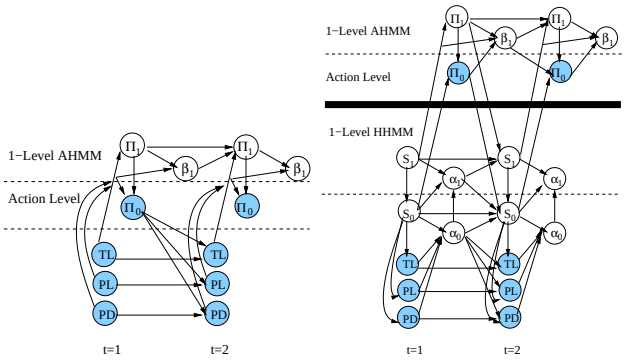
We generated 1000 training sequences from an hierarchical RL mentor trained in the *Taxi* domain using SMDP Q-learning over hand-coded policies. Each sequence is the trajectory of states visited and actions taken in one episode as the RL mentor uses its learned policy to reach the goal. Examples were of variable lengths. A learning rate of $\alpha = 0.1$, an exploration rate of $\epsilon = 0.01$, and a discount rate of $\gamma = 0.9$ were used. Bayes Net Toolbox (Murphy, 2001) was used to implement and train the mAHMM and DAN models in Figure 3 using Expectation-Maximization (Dempster et al., 1977). All distributions were multinomials and except for the β_1, α_0, α_1, Π_1, S_0, and S_1 distributions, were initialized randomly. For the *Taxi* data we set $|S_1| = 5$, $|S_0| = 25$, $|TL| = 25$, $|PL| = 5$, $|PD| = 5$, $|\Pi_1| = 6$, $|\Pi_0| = 6$, $|\alpha_0| = 2$, $|\alpha_1| = 2$, and $|\beta_1| = 2$.

For all experiments, higher level states and policies were biased to change more slowly than lower level states and policies; e.g., the floor you are on should not change more frequently than the room you are in. This was done by initializing the β_1, α_0, α_1, Π_1, S_0, and S_1 distributions as follows, where $i = 0, 1$ and $O_S^t = \{TL^t, PL^t, PD^t\}$.

$$P(\beta_1^t \mid \Pi_0^t, \Pi_1^t, S_1^t) = \begin{cases} 0.95 & \text{if } \beta_1^t = \text{continue} \\ 0.05 & \text{otherwise} \end{cases}$$

$$P(\alpha_0^t \mid S_0^t, O_S^t) = \begin{cases} 0.95 & \text{if } \alpha_0^t = \text{continue} \\ 0.05 & \text{otherwise} \end{cases}$$

$$P(\alpha_1^t \mid \alpha_0^t, S_0^t, S_1^t) = \begin{cases} 1 & \text{if } \alpha_1^t = \alpha_0^t = \text{continue} \\ 0.5 & \text{if } \alpha_1^t = \text{continue and} \\ & \quad \alpha_0^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$



(a) mAHMM (b) DAN

Figure 3. (a) 1-Level mAHMM and (b) 1-Level DAN for Taxi domain; shaded nodes are observed. Note that Π_0 represents actions.

$$P(\Pi_1^{t+1} \mid \Pi_1^t, S_1^t, \beta_1^t)$$

$$= \begin{cases} 1 & \text{if } \beta_1^t = \text{continue and } \Pi_1^{t+1} = \Pi_1^t \\ 1/|\Pi_1| & \text{if } \beta_1^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

$$P(S_0^{t+1} \mid \Pi_0^t, S_0^t, S_1^{t+1}, \alpha_0^t, \alpha_1^t)$$

$$= \begin{cases} 1 & \text{if } \alpha_0^t = \text{continue and } S_0^{t+1} = S_0^t \\ 1/|S_0| & \text{if } \alpha_0^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

$$P(S_1^{t+1} \mid \Pi_1^t, S_1^t, \alpha_1^t)$$

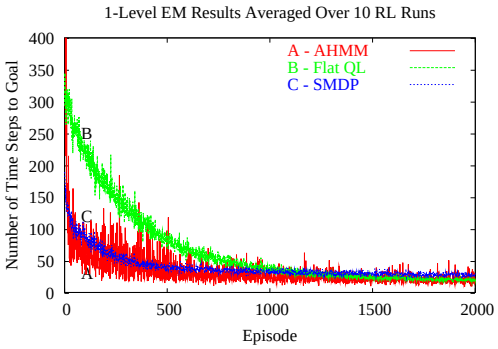
$$= \begin{cases} 1 & \text{if } \alpha_1^t = \text{continue and } S_1^{t+1} = S_1^t \\ 1/|S_1| & \text{if } \alpha_1^t = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

Given the trained mAHMM, policies were extracted and improved using SMDP Q-learning as follows. Once an option (policy) was chosen using an ϵ-greedy exploration strategy, the learned probability distribution $P(\Pi_0|\Pi_1, TL, PL, PD)$ from the mAHMM was used to probabilistically select an action, π_0. Given the trained DAN, policies were again extracted and improved using SMDP Q-learning. However, in order to use the learned probability distributions, we must first compute the most likely abstract state, s_0, with,

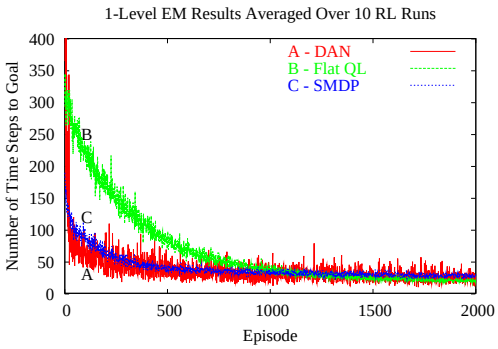
$$P(S_0|TL, PL, PD) =$$

$$\frac{\sum_{S_1} P(TL|S_0)P(PL|S_0)P(PD|S_0)P(S_1)P(S_0|S_1)}{\sum_{S_0,S_1} P(TL|S_0)P(PL|S_0)P(PD|S_0)P(S_1)P(S_0|S_1)}$$

Then given the abstract policy π_1 that was selected using the ϵ-greedy strategy, we can select an action π_0 directly from the conditional probability distribution,



(a) mAHMM



(b) DAN

Figure 4. (a) 1-Level mAHMM results. (b) 1-Level DAN results.

$P(\Pi_0|\Pi_1 = \pi_1, S_0 = s_0)$. Other approaches could be used as well: for instance the full machinery of inference could be used to ascertain how the selected action will affect the predicted distributions over future states. We note that for both the mAHMM and DAN, we permitted options (policies) to be interrupted during SMDP Q-learning. This prevented looping behaviour due to bad policies.

3.3. Results

Figure 4 compares how well the learned mAHMM and DAN policies do against the hand-coded policies and a flat Q-learning agent. What Figure 4 shows is that within about the first fifty timesteps, the SMDP Q-learning agent using the learned policies does as well or better than the SMDP Q-learning agent using the hand-coded policies. We note that the performance of the agent is slightly noisier when it uses the mAHMM rather than the DAN policies.

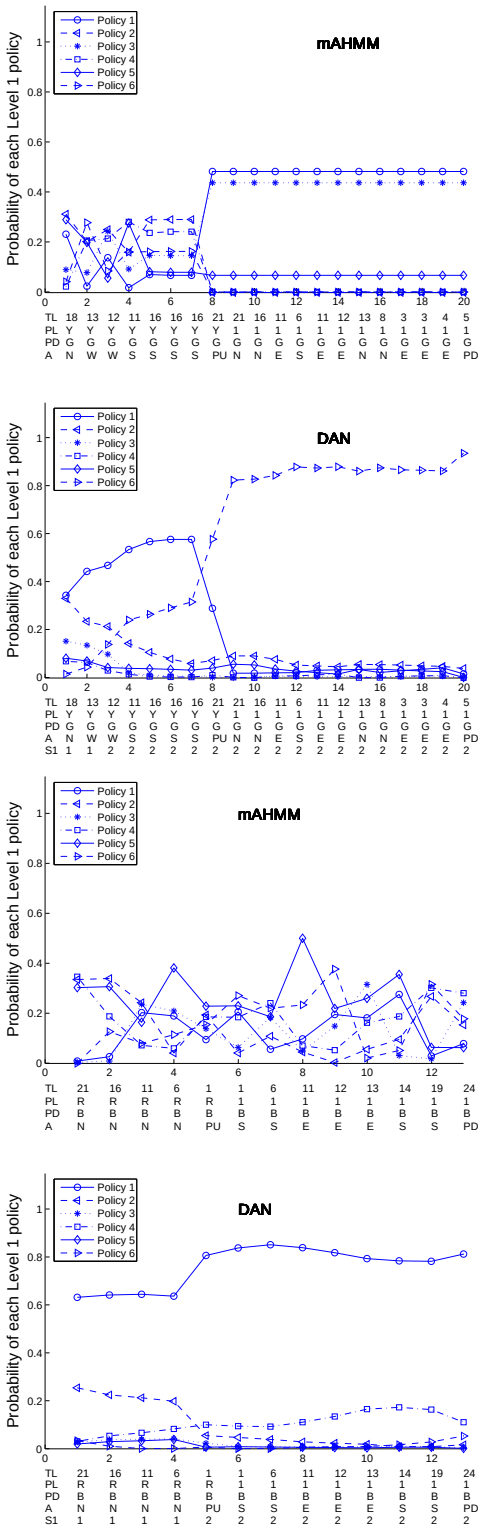


Figure 5. Level 1 policies and states for two sample sequences from the *Taxi* data. The TL, PL, and PD state values and Π_0 actions are shown for both models; the S_0 abstract states are shown for the DAN.

Figure 5 shows the probability of each level 1 policy and state for two sample Taxi sequences for the mAHMM and DAN. What we see from Figure 5 is that the mAHMM has difficulty identifying a single most likely policy with high probability, while the DAN model is able to identify a unique most likely policy with high probability. As shown in Figure 5, the mAHMM performs particularly poorly on the second sequence, never identifying a single policy as most likely for more than a couple of timesteps. Note that while the mAHMM has difficulty identifying a single policy as most likely, this is not a consequence of the policies themselves being poorly learned. Rather, because the mAHMM has learned every policy over the entire state space (due to the structure of the model), all policies are equally good, so any can be used. In essence only a single global policy has been learned. The problem with this is that it cannot be reused, unlike the more local policies learned by the DAN. In particular we see from the DAN graphs in Figure 5 that Policy 1 is used for part or all of both sequences.

4. Conclusions

We have presented a general method for automating hierarchical reinforcement learning. The first phase of our method trains a hierarchical graphical model; the second phase uses the learned policies in an hierarchical reinforcement learning agent. Assuming the graphical model in the first phase encodes the appropriate policy structure, other hierarchical reinforcement learning methods besides SMDP Q-learning can be used for the second phase. In future work we would like to incorporate both phases into an actor-critic architecture. The main disadvantages to our approach are the cost of Expectation-Maximization and having to specify the number of levels and the number of parameters within each level. In future work we plan to explore methods for approximate inference and model selection as applied to dynamic abstraction networks.

Acknowledgments

This work was supported in part by the National Science Foundation under grant ECS-0218125. Any opinons, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Victoria Manfredi was supported by a National Science Foundation graduate research fellowship.

References

Abbeel, P., & Ng, A. (2004). Apprenticeship learning via inverse reinforcement learning. *ICML'04* (pp. 506–513).

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Special Issue on Reinforcement Learning, Discrete Event Systems Journal*, *13*, 41–77.

Bui, H., Venkatesh, S., & West, G. (2002). Policy recognition in the abstract hidden Markov model. *Journal of Artificial Intelligence Research*, *17*, 451–499.

Şimşek, O., & Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. *ICML'04* (pp. 751–758).

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*, 1–38.

Dietterich, T. G. (2000). Hierarchical reinforcment learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, *32*, 41–62.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. *ICML'02* (pp. 243–250).

Manfredi, V., & Mahadevan, S. (2005). Dynamic abstraction networks. *Technical Report 05-33*. Dept of Computer Science, U of Massachusetts Amherst.

Mannor, S., Menache, I., Hoze, A., & Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. *ICML'04* (pp. 751–758).

McGovern, A., & Barto, A. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. *ICML'01* (pp. 361–368).

Murphy, K. (2001). The Bayes net toolbox for Matlab. *Computing Science and Statistics*, *33*.

Murphy, K., & Paskin, M. (2001). Linear time inference in hierarchical hmms. *NIPS'01*.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *NIPS'98*.

Precup, D. (2000). *Temporal abstraction in reinforcement learning*. Doctoral dissertation, University of Massachusetts, Amherst, Department of Computer Science.

Price, B., & Boutilier, C. (2003). Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, *19*, 569–629.

Samejima, K., Doya, K., Ueda, Y., & Kumura, M. (2004). Estimating internal variables and parameters of a learning agent by a particle filter. *NIPS'04*.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Multi-Agent Shared Hierarchy Reinforcement Learning

Neville Mehta MEHTANE@EECS.OREGONSTATE.EDU
Prasad Tadepalli TADEPALL@EECS.OREGONSTATE.EDU
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA.

Abstract

Hierarchical reinforcement learning facilitates faster learning by structuring the policy space, encouraging reuse of subtasks in different contexts, and enabling more effective state abstraction. In this paper, we explore another source of power of hierarchies, namely facilitating sharing of subtask value functions across multiple agents. We show that, when combined with suitable coordination information, this approach can significantly speed up learning and make it more scalable with the number of agents. We introduce the multi-agent shared hierarchy (MASH) framework, which generalizes the MAXQ framework and allows selectively sharing subtask value functions across agents. We develop a model-based average-reward reinforcement learning algorithm for the MASH framework and show its effectiveness with empirical results in a multi-agent taxi domain.

1. Introduction

Reinforcement Learning (RL) is a general learning paradigm in which an agent learns a behavioral policy through interaction with an unknown, stochastic environment (Sutton & Barto, 1998). The agent is capable of taking certain primitive (atomic) actions, and sensing the state of the environment. Recently, there has been a lot of work on *hierarchical reinforcement learning* (HRL) where the state space is hierarchically structured, and composite actions, also known as macros or subtasks, are made available to the agent (Barto & Mahadevan, 2003). One of the frameworks for HRL is based on *options*, which are mini-policies that operate over multiple steps. Options reduce the number of decision points needed to reach the goal since the decisions within the option are fixed a priori (Sutton et al., 1999). *Hierarchies of Abstract Machines* (HAMs) also reduce the number of decisions by representing the policy in the form of a finite state machine with a few non-deterministic choice points (Parr & Russell, 1998); *ALisp* generalizes this idea to arbitrary Lisp programs with choice points (Andre & Russell, 2002). In the *MAXQ* framework, the tasks are organized hierarchically where the only decisions involved are calls to its subtasks (Dietterich, 2000). HRL methods facilitate faster learning by constraining and structuring the space of policies, encouraging the reuse of subtasks in different tasks, and enabling effective task-specific state abstraction.

In this paper, we explore yet another source of power of hierarchies, which is unutilized so far, namely facilitating the sharing of subtask value functions across multiple agents. In previous work, sharing value functions across multiple agents was found to be effective in a non-hierarchical setting when the agents were homogeneous (Tan, 1993). However, a monolithic value function is not necessarily sharable across non-homogeneous agents, since each agent may have its own reward functions. Hierarchies divide the task into smaller subtasks, which makes them more amenable for sharing, since the rewards within a subtask are more likely to be the same across different agents. Thus, if a robot has learned some useful navigational knowledge, e.g., a map for a region, it is much cheaper to communicate it to other robots, rather than having all robots learn this independently. Hierarchies make it possible to share the maps across different robots without also sharing the lower level controls or higher level task knowledge such as delivering mail. The packaging of scientific knowledge in short modular research papers has a similar underlying reason, namely high reusability of such knowledge and the low cost of its communication compared to its discovery.

The rest of the paper is organized as follows: Section 2 explains the theoretical underpinnings of HRL using an example domain. Section 3 generalizes the single-agent HRL framework and the corresponding learning algorithm to the MASH framework. Section 4 evaluates the MASH algorithm in multiple domains, and Section 5 concludes with a discussion of future work.

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

2. Hierarchical Reinforcement Learning

Research in HRL is based on the formalism of *Semi-Markov Decision Processes* (SMDPs), which is an extension of the Markov Decision Process (MDP) formalism. SMDPs allow for temporally extended actions, i.e. actions that can take variable amounts of time as opposed to a fixed interval. Action selections are made at distinct epochs in time, and the state of the system may continue changing during the action. An SMDP is defined as a quintuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{T} \rangle$, where $\mathcal{S}$ is a finite set of states (the state space), $\mathcal{A}$ is a finite set of actions (the action space), $\mathcal{P}$ is the transition probability function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{N} \rightarrow [0, 1]$, $\mathcal{R}$ is the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$, and $\mathcal{T}$ is the transition time function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{N}$. The transition probability function $\mathcal{P}(s', N|s, a)$ is the probability of transitioning to state s' in N time steps, given that action a is taken in state s. The reward function $\mathcal{R}(s'|s, a)$ is the real-valued reward for taking action a in state s and reaching state s'. The transition time function $\mathcal{T}(s'|s, a)$ is the completion time for taking action a in state s and reaching state s'. A policy π is a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$, defining what action the agent takes in any given state.

2.1. Average Reward Learning

If the agent starts in state s and follows policy π, then the *average reward* or *gain* ρ^π of π with respect to s is defined as the ratio of the expected total reward in N steps to the expected total time for N steps as N tends to infinity.

Given an SMDP and a policy π with a gain ρ^π, the *average-adjusted reward* of taking an action a in state s is defined as $r(s, a) - \rho^\pi t(s, a)$, where $r(s, a)$ and $t(s, a)$ are the expected reward and execution time respectively for taking action a in state s. This represents the expected excess reward over what is expected in the duration of the action based on the current average reward. The limit of the total expected average-adjusted reward starting from state s and following policy π is called its *bias* and denoted by $h^\pi(s)$. Under some reasonable conditions on the SMDP structure, there exist a scalar ρ and a real-valued function $h = h^{\pi^*}$ that satisfy the following Bellman equation:

$$h(s) = \max_{a \in \mathcal{A}(s)} \left\{ r(s, a) - \rho t(s, a) + \sum_{s' \in \mathcal{S}} \mathrm{P}(s'|s, a) \cdot h(s') \right\} \quad (1)$$

The policy π^* that selects actions to maximize the right-hand side of Equation 1 attains the optimal gain $\rho^{\pi^*} \geq \rho^\pi$ over all policies π.

2.2. Multi-Agent Taxi Domain

We introduce a multi-agent taxi domain here to help illustrate the concepts presented in this paper. This is a grid world, shown in Figure 1, where the agents are taxis (the squares located on the grid cells) that shuttle passengers (solid circles) from one of four marked cells (labeled 1, 2, 3, and 4) to their intended destinations (again, one of the four marked cells). At any time, there can be at most one passenger waiting at each of these special sites. The generation of passengers can be controlled in different ways, including those based on an arrival probability. The passengers may also depart stochastically from the sites without being transported at all. A full state description of this domain for n agents has 25^n values to indicate the taxi *locations*, 5^n values indicating the *statuses* of all the taxis (empty or an onboard passenger for a particular destination), and similarly 5^4 values indicating the *statuses* of the special sites. Hence, the size of the state space $|\mathcal{S}| = 25^n \times 5^n \times 5^4 = 5^{3n+4}$. The primitive actions available to each agent are moving one cell to the `north`, `south`, `east`, and `west`, `pickup` a passenger, `dropoff` the passenger, and `wait` (no-op). The `wait` action allows the agent to idle when there are no passengers currently waiting to be transported. Every action has a 95% probability of success, otherwise it has no effect. There is a default reward of -0.1 for every action, and a reward of $+100$ for a successful drop-off.

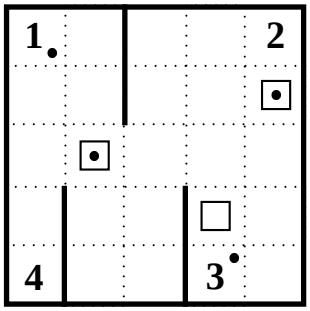


Figure 1. Multi-agent taxi domain. The square boxes represent taxis, and 1, 2, 3, 4 are passenger pickup/drop-off sites. The dots represent passengers.

2.3. HARL Framework

In this section, we describe the Hierarchical Average-reward Reinforcement Learning (HARL) framework, which is an extension of the MAXQ framework (Seri & Tadepalli, 2002; Dietterich, 2000).

In the HARL framework, the original MDP $\mathcal{M}$ is split into sub-SMDPs $\{\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n\}$, where each sub-SMDP represents a subtask (composite or primitive). Solving the root task $\mathcal{M}_0$ solves the entire MDP $\mathcal{M}$. The composite subtasks are primarily designed around subgoals, that is states or regions of the state space, such that reaching those states or regions facilitates achieving the principal goals of the original MDP. The task hierarchy is representable as a

directed acyclic graph known as the *task graph* that represents the task-subtask relationships (e.g., see Figure 2).

For the taxi domain, we create 4 composite subtasks: `Root` represents the overall task; `Get(`l`)` represents the subtask of going to a waiting passenger's location l and picking him up; `Put` represents the subtask of dropping off an onboard passenger at his intended destination; `Goto(`l`)` represents the subtask of going to one of the 4 special locations from anywhere in the grid.

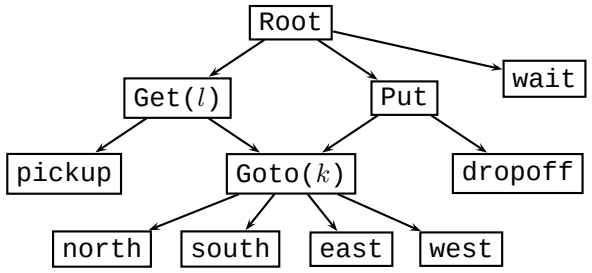


Figure 2. Task graph for the multi-agent taxi domain. Each task calls one of its subtasks based on the current state.

Each subtask $\mathcal{M}_i$ is defined by the tuple $\langle \mathrm{B}_i, \mathcal{A}_i, \mathrm{G}_i \rangle$:

- **State Abstraction** B_i: A function that selects a subset of the original state variables to comprise an abstracted state space sufficient for $\mathcal{M}_i$'s value function to represent the local policy. For example, `Goto(`l`)`'s abstracted state space is composed only of the agent's $location$ variable.
- **Actions** $\mathcal{A}_i$: The set of subtasks that can be called by $\mathcal{M}_i$. For example, `Root` can call either `Get(`l`)`, `Put`, or `wait`.
- **Termination predicate** G_i: The predicate that partitions the subtask's abstracted state space into active and terminated states. When $\mathcal{M}_i$ is terminated, control returns back to the calling subtask. The probability of the eventual termination of a subtask (other than the root task) is 1. For example, `Put` is terminated if there is no onboard passenger.

A subtask is *applicable* iff it is not terminated. All primitive actions of the original MDP are represented as primitive subtasks (leaf nodes) in the task hierarchy. Primitive subtasks have no explicit termination condition (they are always applicable), and control returns to the parent task immediately after their execution. A local policy π_i for the subtask $\mathcal{M}_i$ is a mapping from the states abstracted by B_i to the child tasks of $\mathcal{M}_i$. A hierarchical policy π for the overall task is an assignment of a local policy π_i to each sub-MDP M_i.

To maximize the modularity and reuse of local policies, they have to be recursively gain-optimal. A policy π is *recursively gain-optimal* if the policy π_i at each subtask $\mathcal{M}_i$ maximizes its total expected average-adjusted reward with respect to the gain of the overall hierarchical policy, given the fixed policies of π_i's descendants.

The value function decomposition for a recursively gain-optimal policy satisfies the following set of Bellman equations:

$$\begin{aligned} h_i(s) &= r(s) - \rho \cdot t(s), \quad \text{if } i \text{ is a primitive subtask} \\ &= 0, \quad \text{if } s \text{ is a terminal/goal state for } i\text{; otherwise} \\ &= \max_{a \in \mathcal{A}_i(s)} \left\{ h_a(\mathrm{B}_a(s)) + \sum_{s' \in \mathcal{S}} \mathrm{P}(s'|s,a) \cdot h_i(s') \right\} \end{aligned}$$

The terms $h_i(s)$ and $h_a(s)$ represent the expected average-adjusted rewards of state s during subtasks i and a respectively. If the state abstractions are sound, $h_a(\mathrm{B}_a(s)) = h_a(s)$. The third case of the above equation is derived from Equation 1 by substituting $h_a(s)$ for $r(s,a) - \rho t(s,a)$, which is justified by the definition of $h_a(s)$.

3. MASH Learning

Multi-agent RL is more challenging than single-agent RL because of two complementary reasons. Treating the multiple agents as a single agent increases the state and action spaces exponentially. On the other hand, treating the other agents as part of the environment makes the environment non-stationary and non-Markovian (Mataric, 1997). In this section, we introduce our MASH learning framework in three stages.

3.1. Multi-Agent HH

The most immediate extension of the HARL framework to the multi-agent case is to design a task hierarchy $\mathcal{H}$ for a single agent and then replicate $\mathcal{H}$ for each agent (replacing all agent-related variables appropriately). Some variables that are global to all agents, e.g., the statuses of taxi stands, but some variables are local to each agent, e.g., its own location. Each agent operates alone (selfishly) in this scheme, has a separate task stack, and learns its own value functions and average rewards. Although, there is no explicit coordination, weaker implicit coordination emerges because the learned models within each agent's hierarchy reflect the effects of the other agents on the global state variables. For instance, keeping track of the probabilities of passengers disappearing from the taxi stands allows an agent to detect the effects of the other agents.

3.2. Multi-Agent HH-Coord

A more sophisticated extension to the method described above, pursued by Makar et al. (2001), is to include some kind of explicit coordination. The first stab at including coordination is to have the complete joint state space at all the root tasks, i.e., including all the agent-based and agent-independent state variables in the state abstraction of the root tasks. However, we could approximate the state variables of the other agents by using synthesized variables based on the agents' task stacks. Makar et al. (2001) selected the level one up from the bottom of the task stack as a good high-level approximation to the exact state of an agent. To design a task hierarchy with coordination for an agent x, we add the synthetic coordination variables c that correspond to features of all agents other than x into the root task's state abstraction, and we have the following Bellman backup at the root task:

$$h(s,c) \leftarrow max_a \left\{ h_a(s) + \sum_{s',c'} \mathrm{P}(s',c'|s,c,a) \cdot h(s',c') \right\}$$

However, with this state description, the root task is not strictly an SMDP because the c variables of the modified state space are influenced by the policies of the agents, making it non-Markovian and non-stationary. We all this framework HH-Coord.

If we have n agents and the root task has m subtasks, then the root task's state space grows by a factor of $(m+1)^{n-1}$ because every other agent could either be executing one of the root task's children (m values) or not (1 value). Noting that the identities of the other agents are irrelevant, the root task's state space increases only by a factor of $\binom{(n-1)+(m+1)-1}{n-1}$.

3.3. MASH Framework

We now describe the MASH framework, which allows us to selectively share subtasks between agents[1]. The compelling justification for this is that all agents are interacting with the same environment and every agent sees an isomorphic view of the world involving its own state variables, the agent-independent variables, and the coordination information from the other agents. This allows for an emphasis on subtask sharing across agents. MASH extends HH-Coord by recombining the separate agent hierarchies and value functions back down into one.

Although the task hierarchy is shared, every agent has its own independent task stack, and keeps track of its own global average reward within the system. This allows for asynchronous decision-making by the agents. The agents pool their experiences together to more efficiently learn the value function of every subtask. For instance, for the `Goto(`l`)` task, the more agents we have moving around, the quicker this subtask is learned.

Sharing the task hierarchy prevents the emergence of implicit coordination (as in multi-agent HH) because each agent uses exactly the same value function. For example, all taxis would rush to pick up the same passenger if they are at the same location. Hence, the MASH framework needs the explicit coordination scheme of HH-Coord. Our framework allows the specification of this coordination information in terms of synthetic features that refer to stack variables of other agents.

Parameterization is a powerful mechanism for incorporating prior information into the task hierarchy, and it manifests itself in different ways. First, it can help dynamically adjust the admissibility of subtasks. For instance, `Put` has only one `Goto` subtask that is admissible based on the passenger's destination. Without parameterization, the `Put` subtask would have to learn which ground instance of the `Goto` subtask to select for the passenger. Second, state abstraction and termination conditions are parameterized by the agent's identity and the subtask parameters. For example, Get(l) subtask's abstraction involves only looking at the site$_l$ variable and the agent's location.

3.4. MASH Algorithm

Algorithm 1 specifies the pseudo-code for the MASH learning algorithm. The code is presented from an agent's standpoint. Every agent in the domain executes this code concurrently and asynchronously. The actual execution is a little more complicated than the standard stack-based recursive execution shown here. The algorithm always scans the entire stack for terminated subtasks, starting at the root level. When the first terminated subtask is found, it is popped off along with all of its descendants.

State s is the world state, and every subtask maps s to its abstracted state $\tilde{s}$ using the abstraction function B. For a coordinating subtask, B also encodes the coordination variables (lines 2–3). Unlike the HH algorithm, in MASH, the child task is not restricted to only further abstract from the parent task's abstracted state space. Lines 8–13 are the updates performed at the primitive subtasks, including learning of the reward and time models. For updating the h value, we use $\bar{\rho}$ which is the mean of the agents' separate ρ values. The exploration strategy used is ε-greedy, that is an exploratory action is chosen with a fixed probability (8% in all our experiments), otherwise the greedy action is chosen (lines 16–17). The algorithm is called recursively on the selected subtask a, and the resulting state s' (when the control returns to the calling task) is observed and appro-

[1] Although the MASH framework can allow selective sharing of tasks across agents, this paper only considers the case where all tasks are shared.

Algorithm 1 Multi-Agent Shared Hierarchy Learning Algorithm.

MASH (agent x, task i, state s)

1 **if** (Coordinating-task(i))
2 $c \leftarrow$ encoded coordination information
3 $\tilde{s} \leftarrow \mathrm{B}_i(x, s, c)$
4 **else**
5 $\tilde{s} \leftarrow \mathrm{B}_i(x, s)$
6 **end**
7 **if** (i is primitive)
8 Execute i; observe reward r, elapsed time t, next state s'
9 $\mathrm{N}_i(\tilde{s}) \leftarrow \mathrm{N}_i(\tilde{s}) + 1$
10 $\mathrm{R}_i(\tilde{s}) \leftarrow \mathrm{R}_i(\tilde{s}) + (r - \mathrm{R}_i(\tilde{s}))/\mathrm{N}_i(\tilde{s})$ // Reward model
11 $\mathrm{T}_i(\tilde{s}) \leftarrow \mathrm{T}_i(\tilde{s}) + (t - \mathrm{T}_i(\tilde{s}))/\mathrm{N}_i(\tilde{s})$ // Time model
12 $\overline{\rho} \leftarrow \underset{y \in \text{Agents}}{\text{mean}} \{\rho_y\}$
13 $\mathrm{H}_i(\tilde{s}) \leftarrow \mathrm{R}_i(\tilde{s}) - \overline{\rho} \cdot \mathrm{T}_i(\tilde{s})$ // Primitive H value
14 **else**
15 **while** (**not** (Terminated($i, \tilde{s}$)))
16 Choose a, an exploratory subtask or greedily as:
17 $a \leftarrow \underset{a \in \mathrm{A}_i(\tilde{s})}{\text{argmax}} \left\{ \mathrm{H}_a(\mathrm{B}_a(x, s)) + \sum_{u \in \tilde{\mathcal{S}}_i} \mathrm{P}_i(u|\tilde{s}, a)\, \mathrm{H}_i(u) \right\}$
18 $s' \leftarrow \mathrm{MASH}(x, a, s)$
19 **if** (Coordinating-task(i))
20 $\tilde{s}' \leftarrow \mathrm{B}_i(x, s', c')$ // $c' \leftarrow$ current coordination
21 **else**
22 $\tilde{s}' \leftarrow \mathrm{B}_i(x, s')$
23 **end**

// Update Transition probability model
24 $\mathrm{N}_i(\tilde{s}, a) \leftarrow \mathrm{N}_i(\tilde{s}, a) + 1$
25 $\mathrm{N}_i(\tilde{s}, a, \tilde{s}') \leftarrow \mathrm{N}_i(\tilde{s}, a, \tilde{s}') + 1$
26 $\mathrm{P}_i(\tilde{s}'|\tilde{s}, a) \leftarrow \mathrm{N}_i(\tilde{s}, a, \tilde{s}')/\mathrm{N}_i(\tilde{s}, a)$

27 **if** ($i \neq$ Root task)
// Update Time model
28 $g \leftarrow \underset{b \in \mathrm{A}_i(\tilde{s})}{\text{argmax}} \left\{ \mathrm{H}_b(\mathrm{B}_b(x, s)) + \sum_{u \in \tilde{\mathcal{S}}_i} \mathrm{P}_i(u|\tilde{s}, b)\, \mathrm{H}_i(u) \right\}$
29 $\mathrm{T}_i(\tilde{s}) \leftarrow \mathrm{T}_g(\mathrm{B}_g(x, s)) + \sum_{u \in \tilde{\mathcal{S}}_i} \mathrm{P}_i(u|\tilde{s}, g) \cdot \mathrm{T}_i(u)$
30 **else if** (a was greedily selected)
// Rho update
31 $\texttt{avg-reward}_x \overset{\alpha_x}{\leftarrow} \mathrm{H}_a(\mathrm{B}_a(x, s)) + \rho_x \cdot \mathrm{T}_a(\mathrm{B}_a(x, s)) + \mathrm{H}_i(\tilde{s}') - \mathrm{H}_i(\tilde{s})$
32 $\texttt{avg-time}_x \overset{\alpha_x}{\leftarrow} \mathrm{T}_a(\mathrm{B}_a(x, s))$
33 $\alpha_x \leftarrow \alpha_x/(\alpha_x + 1)$
34 $\rho_x \leftarrow \texttt{avg-reward}_x/\texttt{avg-time}_x$
35 **end**
36 $\mathrm{H}_i(\tilde{s}) \leftarrow \max_{a \in \mathrm{A}_i(\tilde{s})} \left\{ \mathrm{H}_a(\mathrm{B}_a(x, s)) + \sum_{u \in \tilde{\mathcal{S}}_i} \mathrm{P}_i(u|\tilde{s}, a)\, \mathrm{H}_i(u) \right\}$

// Transition to the next state
37 $s \leftarrow s'$; $\tilde{s} \leftarrow \tilde{s}'$
38 **end**
39 $\mathrm{H}_i(\tilde{s}) \leftarrow 0$ // H value for all terminal states = 0
40 **end**
41 **return** s'

priately abstracted (lines 18–23). The composite subtask then updates its transition probability model (lines 24–26).

Lines 27–35 are responsible for the updating the average reward ρ_x for agent x. In the previously published HH algorithm, this was done at the primitive task level only when all the ancestor tasks were selected greedily. However, this resulted in the h values of the root task growing linearly with time. The source of this problem was that the HH algorithm did not take into account the effect of exploratory actions on the ρ update. We modified the current algorithm to update ρ at the root level (whenever it selects a greedy action) using the H learning update rule, which does take exploration into account (lines 31–34). In line 31, since $\mathrm{H}_a(s) = \mathrm{R}(s, a) - \rho \mathrm{T}_a(s)$, the expected reward of executing the root's child task a is $\mathrm{R}(s, a) = \mathrm{H}_a(s) + \rho \mathrm{T}_a(s)$; the expression $\mathrm{H}_i(s') - \mathrm{H}_i(s)$ nullifies the effects of the exploratory actions. In line 32, the right-hand side is the time taken to optimally execute subtask a in state s. Finally, in line 34, ρ_x is evaluated by dividing $\texttt{avg-reward}_x$ by $\texttt{avg-time}_x$. With this new ρ update, $\bar{\rho}$ converges to the correct optimal value (confirmed analytically), and the h values at the root task do not diverge as before. However, this fix entails keeping track of the optimal execution times for all the subtasks (except the root task, which is never-ending). Lines 28–29 learn such a time model for the composite subtasks. Line 36 updates the h value of the composite task; the h value of all terminal states equals zero (line 39).

4. Experimental Results

In this section, we describe the results on a modified version of the taxi domain.

Random passenger generation does not require a lot of coordination between agents because even when two agents try to pick up a passenger from one site, the unsuccessful agent just has to wait a few time steps till the next passenger is generated. To necessitate coordination, the passenger generation is as follows: a destination site is picked and passengers are generated for that destination at all but the destination site. No new passengers are generated until all passengers have been dropped off, after which a new destination site is selected, and so on. All results shown here are for this setting of the taxi domain, averaged over 30 runs. Figure 3 shows the performances of the algorithms in the 5×5 3-agent taxi domain. The MASH algorithm learns the fastest, and converges to the optimal policy. The MASH algorithm without coordination converges to a sub-optimal policy. The HH-Coord algorithm, slowed down by the sheer volume of values it must learn, improves very gradually. Figure 4 shows the performances of the algorithms in the 10×10 version of the 3-agent taxi domain, demonstrating that the MASH framework scales well with

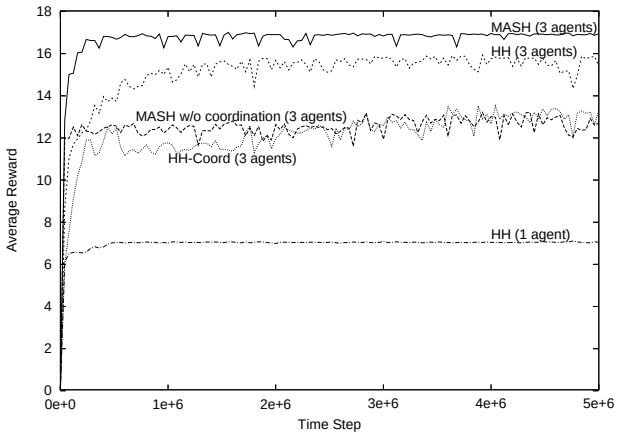


Figure 3. Results in the 5×5 3-agent taxi domain.

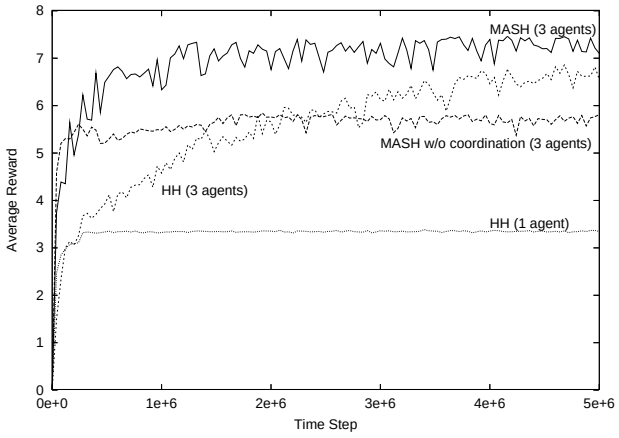


Figure 4. Results in the 10×10 3-agent taxi domain.

the size of the domain. The HH-Coord algorithm could not be run here because of its exorbitant demand for memory.

5. Conclusion and Future Work

We have presented a multi-agent HRL framework called the MASH framework, and its corresponding average-reward learning algorithm. This framework allows agents coordinating in a domain to share their hierarchical value functions to be able to learn more effectively. We have shown that, with suitable coordination information, this framework can greatly boost learning in multi-agent domains. Our framework is extensible to using various kinds of synthesized coordination variables, and to sharing only portions of the task hierarchy (certain subtasks) across multiple agents. Results in the multi-agent taxi domain justify our claims.

Combining the MASH framework with factored action models is an avenue that we are currently researching. Being able to automatically learn the task hierarchy for a given domain is a fundamental issue for future research. This would involve learning the state abstractions, termination conditions, and task-subtask relationships instead of these being provided as prior knowledge to the agents. Moreover, opportunities for sharing these subtasks within the MASH framework would need to be detected automatically.

Acknowledgments

We appreciate the support of NSF under grants ISI-0098050 and ISI-0329278. We would also like to thank all the anonymous reviewers for their valuable comments.

References

Andre, D., & Russell, S. (2002). State Abstraction for Programmable Reinforcement Learning. *Proceedings of the 18th AAAI*.

Barto, A., & Mahadevan, S. (2003). Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Systems*.

Dietterich, T. (2000). Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, *9*, 227–303.

Makar, R., Mahadevan, S., & Ghavamzadeh, M. (2001). Hierarchical Multi-Agent Reinforcement Learning. *Proceedings of the 5th International Conference on Autonomous Agents*.

Mataric, M. (1997). Reinforcement Learning in the Multi-Robot domain. *Autonomous Robots, 4(1)*, 73–83.

Parr, R., & Russell, S. (1998). Reinforcement Learning with Hierarchies of Machines. *Advances in Neural Information Processing Systems* (pp. 1043–1049). MIT Press.

Seri, S., & Tadepalli, P. (2002). Model-based Hierarchical Average Reward Reinforcement Learning. *Proceedings of the 19th International Conference on Machine Learning* (pp. 562–569).

Sutton, R., & Barto, A. (1998). *Reinforcement Learning*. MIT Press.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence, 112(1-2)*, 181–211.

Tan, M. (1993). Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. *Proceedings of the 10th International Conference on Machine Learning* (pp. 330–337).

Convergence of reinforcement learning using a decision tree learner

Jan Ramon JAN.RAMON@CS.KULEUVEN.AC.BE
Department of Computer Science, Katholic University of Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium

Abstract

In this paper, we propose conditions under which Q iteration using decision trees for function approximation is guaranteed to converge to the optimal policy in the limit, using only a storage space linear in the size of the decision tree. We analyze different factors that influence the efficiency of the proposed algorithm, and in particular study the efficiency of different concept languages. We illustrate the approach with some preliminary experiments.

1. Introduction

A large number of reinforcement learning algorithms such as policy iteration, value iteration and Q-learning require a table with one or more entries for each state in the state space. Unfortunately, this becomes infeasible when dealing with large state spaces. A common approach is to use function approximation. However, while the convergence of the table-based algorithms is well studied (Singh et al., 2000), there are only few results on the convergence of approaches using function approximation. State aggregation, which can be seen as a function approximation approach using functions constant over the aggregated states, is known to converge, but there is no guarantee that there will be convergence to the optimal policy.

In this paper we propose a method learning the Q-function that adapts state aggregation in such a way that convergence to the optimal policy in the limit is guaranteed. While we do not prove that the resulting state aggregation is optimal (according to the number of states or some other optimality criterion), in many cases the algorithm will require much less memory than the existing algorithms having convergence guarantees.

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

The main idea is that we refine a state aggregation iteratively by splitting abstract states into smaller abstract states, until an optimal (or sufficiently good) policy is obtained. Essentially, we are using an (online) decision tree learning approach (Chapman & Kaelbling, 1991; Driessens et al., 2001). The crucial point is to show that if a further split is needed to find an optimal policy, then a test will be found that improves the current approximation. Once this has been established, it is straightforward to show convergence to the optimal policy. The contributions of this work can be separated into two parts. In a first part, we will propose a sufficient condition for convergence to the optimal policy. In a second part, we will not only focus on conditions that guarantee convergence, but also on convergence speed and memory usage.

A number of recent research interests provide an additional motivation for this work. In particular, there has been a growing interest in relational reinforcement learning (Tadepalli et al., 2004), where relational languages are used to represent abstractions of states. After an earlier period when several algorithms have been proposed and validated empirically, there is a growing interest in a theoretical base for relational reinforcement learning. This work contributes first by providing an extension to (Kersting & De Raedt, 2004) in that it allows for abstractions being refined over time. Second, it contributes a first step towards a more systematic study of the properties of languages which can be used to describe abstractions, in particular properties needed to be able to converge to the optimal policy. A third motivation comes from the experience of several researchers in the field of Inductive Logic Programming (see e.g. (Sebag, 1997; Srinivasan, 2000)) that a practical solution to keep the huge hypothesis spaces generated by these rich languages tractable, may be found in stochastically sampling from these languages. In fact, while conventional reinforcement learners only explore the state space, this work also explores the concept space.

The proofs of our results and a more elaborate discussion can be found in (Ramon, 2005).

2. Preliminaries

We assume that the agent is operating in a MDP and adopt common notational conventions. A *ground state-action pair* or briefly *(ground) SA-pair* is an element of $\mathcal{S} \times \mathcal{A}$. We will denote ground SA-pairs with lowercase letters. A *SA-aggregation* D is a set of disjoint SA-abstractions $D = \{SA_1, \ldots, SA_N\}$ where each *SA-abstraction* SA_i is a set of state-action pairs (i.e. $SA_i \subseteq \mathcal{S} \times \mathcal{A}$), such that $\cup_{i=1}^{N} SA_i = \mathcal{S} \times \mathcal{A}$ SA-abstractions are denoted with capital letters.

If D is a SA-aggregation and sa is a SA-pair, then we will denote with $D(sa)$ the SA-abstraction $SA \in D$ such that $sa \in SA$. As we assume the state space to be very large, storing knowledge on the basis of individual states is infeasible. Therefore, descriptive languages have been introduced (Kaelbling et al., 2001; Kersting & De Raedt, 2004) that allow to implicitly describe states and actions. In the paper we will represent an SA-aggregation with a (logical) decision tree, such that the SA-abstractions are the leaves of the tree and the ground SA-pairs can be sorted down the tree to determine to which abstraction they belong. This ensures that SA-abstractions do not overlap, which will simplify the proofs of our results.

A concept over a space $\mathcal{U}$ is a subset of $\mathcal{U}$. A concept language (denoted with $\mathcal{L}$) over a space $\mathcal{U}$ is a set of concepts over $\mathcal{U}$. A concept $c \in \mathcal{L}$ covers an element $u \in \mathcal{U}$ iff $u \in c$. The concept language $\mathcal{L}$ will be a parameter to the learning algorithm. For simplicity of explanation, we will assume concept languages (and state-action spaces) to be finite. Nevertheless, when dealing with a particular SA-abstraction SA, we will only consider a sublanguage $\mathcal{L}(SA) \subseteq \mathcal{L}$ of concepts relevant for SA. This means we remove all concepts c which are trivial on SA (either $c \cap SA = SA$ or $c \cap SA = \emptyset$), and we only keep one concept for each equivalence class of concepts c for which the sets $c \cap SA$ are identical.

3. Algorithm

A high level version of our algorithm is shown in Algorithm 1. Our algorithm keeps a current SA-abstraction D at all times, together with a current Q-function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Essentially, it can be seen as a decision tree algorithm as it iteratively tries to split nodes (SA-abstractions) into smaller ones, creating a tree-structured hierarchy of SA-abstractions.

After choosing candidate splits in step 4, the agent explores the world until sufficient (see later) statistics are collected. Different strategies for exploration can be applied.

Algorithm 1 High level algorithm

1: $D \leftarrow \{\mathcal{S} \times \mathcal{A}\}$
2: **repeat**
3: **for all** $SA_i \in D$
4: Choose randomly k concepts $c_{i,j} \in \mathcal{L}(SA_i)$ $(j = 1 \ldots k)$ as candidate split.
5: Explore_and_collect_statistics($D \cup \{SA_i \cap c_{i,j}, SA_i \setminus c_{i,j}\}_{SA_i \in D, 1 \leq j \leq k}$)
6: **for all** $SA_i \in D$
7: **for** $j = 1 \ldots k$
8: Check whether the states in $SA_i \cap c_{i,j}$ and $SA_i \setminus c_{i,j}$ behave different. If so, the candidate is valid.
9: **if** at least one candidate split is valid
10: choose the best split c_i^*.
11: $D \leftarrow D \cup \{SA_i \cap c_i^*, SA_i \setminus c_i^*\} \setminus \{SA_i\}$
12: update Q using the collected statistics.
13: **until** all states ϵ-uniform

Next, the algorithm tries to determine from the collected statistics whether the SA-abstractions SA_i should be refined (split into smaller SA-abstractions) in order to improve the Q-function. If at least one useful split is found among the k trials, the SA-abstraction is split into two. A test is useful if it splits the SA-abstraction into two subsets for which the average Q-value (at this point in the iteration) is sufficiently different. If several candidate splits are acceptable, the SA-abstraction is split according to some heuristic, as usual in decision tree learners.

After splitting SA-abstractions, the algorithm updates its Q functions using the statistics collected during exploration (step 12).

Algorithm 2 shows step 5 of Algorithm 1 in more detail. Algorithm 2 has as parameter a set of concepts C, including the SA-abstractions resulting from the generated candidate splits. The algorithm explores the state-action space for a sufficiently long time. For the discussion of this algorithm, we will denote the visited states with s_i, the actions taken with a_i, the rewards received with R_i. Furthermore, $q'_i = R_i + \gamma \max_{a \in \mathcal{A}} Q(D(s_{i+1}, a))$. The algorithm counts for every concept $c \in C$ how many steps started in a SA-pair belonging to that concept, and makes sums $t(c) = \sum_i \{q'_i | (s_i, a_i) \in c\}$ and counts $n(c) = \#\{i | (s_i, a_i) \in c\}$. Step 12 of Algorithm 1 can then be performed by setting $Q(SA) \leftarrow t(SA)/n(SA)$, for all $SA \in D$.

Notation 1 *When considering the statistics collected during exploration, we will use for any SA-pair* $sa \in \mathcal{S} \times \mathcal{A}$*, the notation* $\tilde{q}(sa) = \sum \{q'_i | s_i = s \wedge$

Algorithm 2 Explore and collect statistics

Require: C : a set of concepts from concept language $\mathcal{L}$

```
for all c ∈ C
    n(c) ← 0
    t(c) ← 0
s_1 ← initial_state
forall i = 1 ... NbSteps
    Choose action a_i
    Perform action a_i, receiving reward R_i and next state s_{i+1}
    q'_i = R_i + γ max_{a∈A} Q(D(s_{i+1}, a))
    for allc ∈ C
        if c(s_i.a_i)
            n(c) ← n(c) + 1
            t(c) ← t(c) + q'_i
```

$a = a_i\}/\#\{i|s_i = s \wedge a = a_i\}$ *for the estimated Q-value of this state-action pair. Note that we do not keep all these numbers in memory, but will use them only for our reasoning. Also, we will use* $\tilde{q}(SA) = \sum\{q'_i|(s_i, a_i) \in SA\}/\#\{i|(s_i, a_i) \in SA\} = \frac{t(SA)}{n(SA)}$ *for the estimated average Q-value in the SA-abstraction* SA.

4. Finding tests until convergence

In this section, we will analyze our algorithm in more detail and formulate some general conditions for convergence.

Definition 1 (uniform abstractions) *Let* $SA \subseteq \mathcal{S} \times \mathcal{A}$ *be a SA-abstraction and let* $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *be a real-valued function on* $\mathcal{S} \times \mathcal{A}$. *Then we say that* SA *is* ϵ*-uniform w.r.t.* f *iff for all* $sa_1, sa_2 \in SA$, $|f(sa_1) - f(sa_2)| < \epsilon$.

Essentially, the algorithm iterates over replacing its current approximation of the Q-function with an approximation of the improvement Q' where

$$Q'(s,a) = \sum_{s' \in \mathcal{S}} p(s,a,s')\big(r(s') + max_{a' \in \mathcal{A}} Q(s',a')\big). \tag{1}$$

Both the old and the new approximations are represented by decision trees. Every internal node that occurs in the old approximation also occurs in the new one. The only difference is that for the new approximation a leaf may be replaced by a leaf with a different prediction, or a leaf may be replaced by a new internal node with two new leaves. In fact, given some current estimate Q, the algorithm splits the state-action space into parts with a more uniform value for Equation (1). If we can guarantee that abstractions will be split as long as they are not uniform with respect to Q', it is not difficult to see that from a certain point no splits will be necessary any more and an iterative application of the Q update will converge to the optimal Q function.

So our algorithm generates a new estimate Q' from a previous estimate Q. In our analysis we will employ three different kinds of quantities. First, there is the quantity $Q'(s,a)$ which is the update we would ideally want to obtain. Second, there is the estimate $\tilde{q}(sa)$ of Q' that is measured in the exploration performed by the algorithm. $\tilde{q}(sa)$ and $Q'(sa)$ may be different as the environment may be non-deterministic and we do not explore for an infinite time. Third, there is the $\tilde{q}(c)$ with $c \subseteq \mathcal{S} \times \mathcal{A}$, which is an average of the function $\tilde{q}$ over an abstraction c. In our algorithm, only the third kind of value is stored in memory in the form of the statistics collected by Algorithm 2. Our task will be to derive from the latter values suitable bounds for the $\tilde{q}(sa)$ and later for $Q'(sa)$. We will first introduce some more notations.

Notation 2 (indicator function) *Let* $\mathcal{L}$ *be a concept language over* $\mathcal{U}$ *and let* $u \in \mathcal{U}$ *and* $c \in \mathcal{L}$. *Then we denote with* 1_c *the indicator function of* c, *defined by* $1_c(r) = 1$ *iff* $r \in c$, *else* $1_c(r) = 0$. *If* $C \subseteq \mathcal{L}$ *is a set of concepts, we will denote with* 1_C *the set of indicator functions* $\{1_c|c \in C\}$.

Notation 3 (boldface) *If* $U = \{u_1, u_2, \ldots, u_N\}$ *is a set and* f *is a function, then we will use the boldface notation* $\mathbf{f}[U]$ *for a column vector* $(f(u_1)\, f(u_2)\, \ldots\, f(u_N))^T$. *For simplicity of notation, vectors and matrices will be indexed with argument names (concepts, SA-pairs, ...) instead of natural numbers, e.g.* $\mathbf{f}[U]_{u_i} = f(u_i)$. *E.g.* $\mathbf{1}[U]$ *is a vector of* $\#U$ *ones. If* $F = \{f_1, f_2, \ldots, f_N\}$ *is a set of functions, we will denote with* $\mathbf{F}[U]$ *the matrix* $[\mathbf{f_1}[U]\ \mathbf{f_2}[U]\ \ldots\ \mathbf{f_N}[U]]$.

So if $C = \{c_1, \ldots, c_N\}$ is a set of concepts for which we collected statistics in Algorithm 2, then we denote with $\tilde{\mathbf{q}}[C]$ the column vector $[\tilde{q}(c_1), \ldots, \tilde{q}(c_N)]^T$, compactly representing the measured averages of $\tilde{q}$ on the part covered by the c_i of the state action space. On the other hand, $\tilde{\mathbf{q}}[SA]$ is the column vector with $\tilde{\mathbf{q}}[SA]_{sa}$ the average observed Q value at sa. While we do not know the values of the latter vector, there is a linear relation between both:

$$\tilde{\mathbf{q}}[C] = K(C, SA).\tilde{\mathbf{q}}[SA] \tag{2}$$

where $K(C, SA)$ is a $\#C \times \#SA$ matrix. $K(C, SA)$ can be decomposed into a product of three matrices:

$$K(C, SA) = \Lambda'.\big(\mathbf{1_C}[SA]\big)^T.\Lambda. \tag{3}$$

Here, $\mathbf{1_C}[SA]$ is a $\#SA \times \#C$ matrix expressing which concept covers which SA-pairs. $\Lambda \in \mathbb{R}^{SA \times SA}$ and $\Lambda' \in \mathbb{R}^{C \times C}$ are diagonal matrices taking care of the correct weighting of the SA-pairs in the statistics of the concepts. In particular, $\Lambda_{sa,sa} = \#\{i|(s_i,a_i) = sa\}/NbSteps$ is the fraction of the time we were in SA-pair sa and $\Lambda'_{c,c} = 1/\sum_{sa \in c} \lambda(sa)$. We can also write Equation (2) as

$$(\tilde{\mathbf{q}} - \bar{\mathbf{q}}_{\mathbf{SA}})[C] = K(C,SA).(\tilde{\mathbf{q}} - \bar{\mathbf{q}}_{\mathbf{SA}})[SA] \quad (4)$$

where the constant $\bar{q}_{SA} = \tilde{q}(SA)$ is the average $\tilde{q}$ value measured in SA, i.e. $(\tilde{\mathbf{q}} - \bar{\mathbf{q}})[SA]$ is a column vector of deviations of the measured Q values at individual SA-pairs from the average over the abstraction SA.

Definition 2 *Let $\mathcal{L}$ be a concept language over a space $\mathcal{U}$, Let $C = (c_1, \ldots, c_N) \in \mathcal{L}^N$ be a vector of N concepts in $\mathcal{L}$. Let U be a set. Then, we say that C is scattering-generating for U or scat-gens(C,U) iff the rank of $\mathbf{1_C}[U]$ is $\#U$.*

The VC dimension of a concept class is the size of the largest set the concept class shatters, i.e. the largest set U such that for every subset $X \in U$, there is a concept c such that $U \cap c = X$. In (Ramon, 2005) we prove that $\mathbb{R}^{\mathcal{L}}$ (the set of linear combinations of concepts of $\mathcal{L}$) is scatters every set $U \subseteq \mathcal{U}$ of size n if an only if $\mathcal{L}$ is scattering generating for every subset $U \subseteq \mathcal{U}$ of size n.

It is clear that if scat-gens$(\mathcal{L}, \mathcal{S} \times \mathcal{A})$ does not hold, there may be a non-uniformity w.r.t. the Q function that we can not detect. Indeed, $rank(\mathbf{1}_{\mathcal{L}}[\mathcal{S} \times \mathcal{A}]) < \#\mathcal{S} \times \mathcal{A}$ would imply that there is a non-zero vector Q_0 (indexed by SA pairs from $\mathcal{S} \times \mathcal{A}$) such that $\mathbf{1}_{\mathcal{L}}[\mathcal{S} \times \mathcal{A}].Q_0 = 0$, and if that would be the Q-function (i.e. $\mathbf{Q}[SA] = Q_0$), there would be no concept that would allow us to detect that some abstractions are non-uniform.

We will now present some relations concerning the error we make by exploring for only a finite time and by using statistics instead of a full table of Q-values. The proof of these results can be found in (Ramon, 2005).

Let σ_R^2 be the maximal variance on $R(s)$ for a particular state s (due to the non-determinacy of the environment) and let $\sigma_Q = \sigma_R/(1-\gamma)$. Assume we visit every SA-pair of a particular SA-abstraction SA at least M times. One can prove the following bound on the sampling error, which is used in the proofs of the following theorems:

$$P\left(\left\|\mathbf{Q}'[SA] - \tilde{\mathbf{q}}[SA]\right\|_\infty \geq z\sigma_Q/\sqrt{M}\right) \leq \#SA.\sqrt{2/\pi z^2}\exp(-z^2/2) \quad (5)$$

Theorem 1 *Assume Algorithm 2 has collected statistics for a set of N concepts $C = \{c_1, c_2, \ldots, c_N\}$ of an SA abstraction SA with $\#SA = n$. Let $\epsilon > 0$. Assume furthermore that*

$$\|(\tilde{\mathbf{q}} - \bar{\mathbf{q}}_{\mathbf{SA}})[C]\|_\infty \geq 2\epsilon. \quad (6)$$

Then with probability at least $1-\delta$, SA is not ϵ-uniform w.r.t. Q', where $\delta = \left(\sqrt{2}\sigma_Q n/\sqrt{\pi}\epsilon\sqrt{M}\right)\exp(-\epsilon^2 M/2\sigma_Q^2)$

The next theorem shows how to ensure that an abstraction for which we do not detect non-uniformity is indeed uniform.

Theorem 2 *Let $\mathcal{L}$ be a concept language over $\mathcal{S} \times \mathcal{A}$ with a probability distribution on it. Let $SA \subseteq \mathcal{S} \times \mathcal{A}$ (with $\#SA = n$) is not ϵ-uniform. Let Algorithm 2 collect statistics for a set of N randomly drawn concepts C and visit every SA-pair at least M times. Assume that with probability at least $1-\delta_2$ it holds that*

$$\beta \leq 4^{-1}\|K(C,SA)^\dagger\|_\infty^{-1} \quad (7)$$

for some $\beta > 0$. Then, with probability $1-\delta_1-\delta_2$, the following holds:

$$\|(\tilde{\mathbf{q}} - \bar{\mathbf{q}}_{\mathbf{SA}})[C)]\|_\infty \geq 2\epsilon\beta \quad (8)$$

where $\delta_1 = 2^{3/2}\pi^{-1/2}n^{3/2}\sigma_Q M^{-1/2}\epsilon^{-1}\exp(-\epsilon^2 M/8n\sigma_Q^2)$

In Equation (7), we are actually relating β to the condition number $\|K(C,SA)\|_\infty.\|K(C,SA)^\dagger\|_\infty$ of the matrix $K(C,SA)$ (the proof of Theorem 1 shows that $\|K(C,SA)\|_\infty = 1$). The condition number is a commonly used measure in matrix algebra.

Theorem 1 states conditions under which we can be arbitrarily sure that an abstraction is not ϵ-uniform and we have found a split. On the other hand, Theorem 2 states conditions such that if an abstraction is not ϵ-uniform, we will with high probability get into a situation where the conditions of Theorem 1 are satisfied (notice that Equations 6 and 8 differ only in the constant β). Therefore, these theorems together provide conditions under which we are guaranteed to detect every non-ϵ-uniformity. Conversely, if an abstraction is $\beta\epsilon$-uniform, we are guaranteed to not find a split (because that would imply that the abstraction is not ϵ-uniform). So if we want to build a decision tree with accuracy ϵ, the splitting is guaranteed to continue as long as the nodes are not ϵ-uniform, and may continue a few levels too far in worst case, but will then stop. In (Ramon, 2005), we use this argument iteratively to prove that the learning algorithm will converge to a Q function with error at most ϵ from the optimal one,

using only storage space linear in the size of the decision tree. In (Ramon, 2005) we prove that Theorems 1 and 2 together ensure that the algorithm converges to a Q function which approximates the optimal one with error bounded by an arbitrary ϵ and that memory linear in the size of the decision tree is needed.

5. Influence of the concept language

So first we will study the influence of the choice of concept language. As argued in section 4, besides sufficient sampling a second requirement for success is that the matrix $(\mathbf{1_C}[SA])^T$ (with C the concepts) is sufficiently well conditioned. In other words, the concept language should be sufficiently rich and provide a range of tests as diverse as possible.

We first list the languages we are considering: $\mathcal{L}_\delta = \left\{\{sa\}|sa \in \mathcal{S}\times\mathcal{A}\right\}$ is the language of all concepts covering exactly one SA-pair. We consider $\mathcal{L}_{f,\wedge\geq} = \left\{\{x|x \geq a\}|a \in \mathcal{S}\times\mathcal{A}\right\}$, where for elements $x, a \in \mathcal{S}\times\mathcal{A}$ with $x = (x_1, \ldots, x_m)$ and $a = (a_1, \ldots, a_m)$ we define $x \geq a \Leftrightarrow x_1 \geq a_1 \wedge x_2 \geq a_2 \wedge \ldots \wedge x_m \geq a_m$. Another factored language is $\mathcal{L}_{f,apr} = \left\{\{x|a \leq x \leq b\}|a, b \in \mathcal{S}\times\mathcal{A}\right\}$. This is the language of all axes-parallel rectangles. If the state-action space is a vector space, one can also use $\mathcal{L}_{f,circ} = \left\{\{x|d(x,a) < d_{max}\}|a \in \mathcal{S}\times\mathcal{A} \wedge d_{max} \in \mathbb{R}\right\}$ where $d(x,a)$ is the Euclidean distance between x and a. Hence, concepts from $\mathcal{L}_{f,circ}$ are spheres. We also consider $\mathcal{L}_{f,hs} = \left\{\{x|x.a \geq b\}|a \in \mathcal{S}\times\mathcal{A} \wedge b \in \mathbb{R}\right\}$, the language of all half-spaces and $\mathcal{L}_{f,lex} = \left\{\{x|x \succeq a\}|a \in \mathcal{S}\times\mathcal{A}\right\}$ (with $\succeq$ the lexicographic order on tuples).

In (Ramon, 2005) we provide a closed form for β for a.o. $\mathcal{L}_\delta$ and $\mathcal{L}_{f,\wedge\geq}$. One of these languages is of theoretical interest but can not be used in practice: $\mathcal{L}_\delta$ is optimally conditioned ($\beta = 1$) but since in each node, only one example is split off, the system will revert to a table-based approach. It is in general good to use a concept language where each concept covers a significant fraction of the state-action space.

Consider a 2D state space of dimension 10×10 with 4 actions in each state. The reward and transition functions are choosen completely randomly (there is no relation between the dynamics and the 2D structure of the world). We computed a suitable value for β for each of the languages $\mathcal{L}_{f,\wedge\geq}$, $\mathcal{L}_{f,lex}$, $\mathcal{L}_{f,circ}$, $\mathcal{L}_{f,hs}$ and $\mathcal{L}_{f,apr}$ by computing condition numbers of $\mathbf{1_C}[\mathcal{S}\times\mathcal{A}]$ for samples of concepts C (we assume equal $\Lambda_{sa,sa}$ for all sa). The results are shown in Table 1. Next, we ran the decision tree learner on this world using the different languages. In this experiment, we only split nodes if the means of the candidate subnodes is more than some constant d. We recorded the size of the learned trees. One can see from Table 1 that languages with a smaller β find more splits with a given difference.

Lang	$\mathcal{L}_{f,\wedge\geq}$	$\mathcal{L}_{f,lex}$	$\mathcal{L}_{f,circ}$	$\mathcal{L}_{f,hs}$	$\mathcal{L}_{f,apr}$	$\mathcal{L}_\delta$
β	121	71	45	36	5	1
Treesize	121	148	156	161	164	n.a.

Table 1. Languages, sampled order of magnitudes for β, and size of the learned tree

6. Influence of learning parameters

Next, we present preliminary experiments illustrating the behavior of our implementation in practice. Our goals are the following. We want to show how the learning results improves as the approximation of Q gets better, and what is the needed storage space. Next, we want to know what is the influence of the number k (see Algorithm 2) of candidate splits that is tried in each exploration phase.

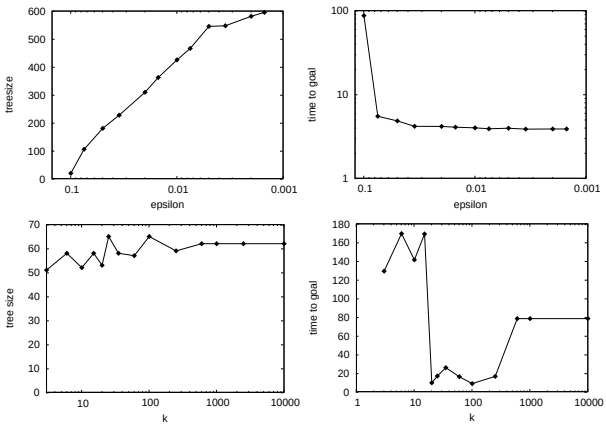


Figure 1. Top: Tree size (left) and time to goal (right) as function of ϵ. Bottom: Tree size (left) and time to goal (right) as function of k.

We consider a simple 2D maze world where the agent can move one step in one the the eight directions (horizontal, vertical, diagonal), at each time point. The world has 10×10 states and there is one goal at $(5, 6)$. There are walls, blocking the way between states (i.e. moving against a wall does not change the position of the agent). The initial state is choosen randomly. An episode ends either after the agent reached the goal or after 1000 steps. The fact that this world is deterministic allows for several optimizations and simplifications. We use concept language $\mathcal{L}_{f,\wedge\geq}$.

We report the average time it takes the agent to reach his goal, rather than reporting the reward which is always 1 after a successful episode. The top of Figure

1 plots tree size and average time to reward against ϵ, where ϵ is the required accuracy. All data points are obtained from an average of 100 tests. After this accuracy is reached for some abstraction, the tree does not refine it any more. One can see that already for a relatively large values ϵ, a rather good performance is reached, while smaller values for ϵ produce much larger trees without improving the performance much more. The bottom of Figure 1 plots the tree size and the average time to goal against the parameter k of Algorithm 2. These values are measured after 100 episodes. The size of the trees produced up to that point do not differ significantly in size. On the other hand, it is interesting to see that there is some optimal value for k. One explanation is that large values for k, meaning many candidate splits, give a stronger incentive to overfitting. On the other hand, smaller values for k let the algorithm take just the first acceptable test, without giving it much chance to compare it to others.

We get similar results in the blocks world (Driessens et al., 2001), but omit them due to space limits.

7. Discussion

We presented an algorithm for approximating Q-functions with decision trees that uses only memory linear in the size of the decision tree. This paper has two main contributions. First, we proved that this method can approximate the optimal Q-function with error smaller than a choosen ϵ, and hence that the algorithm converges in the limit to an optimal policy. Second, we discussed the influence of the concept language used and proposed measures for the power of the concept language and the probability distribution on it. We illustrated this with experiments.

Our bounds do not make assumptions on the shape of the Q-function. If this function satisfies certain conditions, one can probably show much better results. Also, depending on the application, it is easier to represent either the Q function or the policy compactly. We therefore expect that applying our approach to policy iteration (instead of Q-iteration) would extend the range of applications where trees can be used.

In practice, convergence may happen faster than our theoretical results guarantee. This depends mainly on the application at hand. While in theory it can take a long time before stochastically choosing candidate concepts guarantees that at least one useful concept will be found, other authors have argued before that stochastically trying concepts is valuable (Sebag, 1997; Srinivasan, 2000). The results in this paper could offer inspriation on what distributions are useful for randomly generating concepts.

An interesting open question is whether similar bounds can be proved for supervised decision tree learning. Up to now, research on the concept languages used for supervised decision tree learning has been limited, and the results from this paper are not applicable directly as we adopt the usual reinforcement learning assumption that we can visit all states. Nevertheless, we anticipate that in some cases, one can prove bounds on the time needed to learn in a more general setting.

Acknowledgements

Jan Ramon is a post-doctoral fellow of the Fund for Scientific Research (FWO) of Flanders.

References

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisions. *Proc. of IJCAI'91* (pp. 726–731).

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. *Proc¡ of ECML'01* (pp. 97–108).

Kaelbling, L. P., Oates, T., Gardiol, N. H., & Finney, S. (2001). Learning in worlds with objects. *AAAI Stanford Spring Symposium on Learning Grounded Representations*.

Kersting, K., & De Raedt, L. (2004). Logical markov decision programs and the convergence of logical td(λ). *Proc. of ILP'04*.

Ramon, J. (2005). *On the convergence of relational reinforcement learning using decision trees for function approximation* (Technical Report 2005). ? `http://www.cs.kuleuven.ac.be/ janr/tgconv/`.

Sebag, M. (1997). Distance induction in first order logic. *Proc. of ILP'97* (pp. 264–272).

Singh, S., Jaakkola, T., M.L., L., & Szpesvari, C. (2000). Convergence results for single step on-policy reinforcement-learning algorithms. *Machine Learning, 38*, 287–308.

Srinivasan, A. (2000). *A study of two probabilistic methods for searching large spaces with ILP* (Technical Report PRG-TR-16-00). Oxford University Computing Laboratory.

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinfocement learning: An overview. *ICML-04 Workshop on Relational Reinfocement Learning*.

Simultaneous Learning of Structure and Value in Relational Reinforcement Learning

Scott Sanner SSANNER@CS.TORONTO.EDU
Department of Computer Science, University of Toronto, Toronto, ON M5S 3H5, CANADA

Abstract

We introduce an approach to model-free relational reinforcement learning in finite-horizon, undiscounted domains with a single terminal reward of success or failure. We represent the value function as a relational naive Bayes network and show that both the value (parameters) and structure of this network can be learned efficiently under a minimum description length (MDL) framework. We describe the SVRRL and FAA-SVRRL algorithms for efficiently performing simultaneous structure and value learning and apply FAA-SVRRL to the domain of Backgammon. FAA-SVRRL produces a high-performance agent in very few training games and with little computational effort, thus demonstrating the efficacy of the SVRRL approach for large relational domains.

1. Introduction

The field of relational reinforcement learning (RRL) has emerged in recent years as a major area of focus in the reinforcement learning community (Tadepalli et al., 2004; van Otterlo & Kersting, 2004). While RRL is an attractive approach for learning from delayed reward in a relational state representation, its generality does not come without severe representational and computational drawbacks:

- As the number of ground domain objects and the arity of the relations increase, there is a combinatorial explosion in the number of ground relations that describe a state. This results in an extremely large state space that can quickly become unmanageable, even if there are only a few relations in the problem specification.
- One must carefully decide on the hypothesis space from which a value function or policy may be selected. If too simple of a space is used, the learner may not be able to obtain a good representation of the value function. And if too complex of a space is used, the learner may never be able to find a good representation or obtain enough data to achieve a low-variance estimate of the value function.
- Finding a relational structure for a value function or policy that is optimal for all domain instantiations is extremely computationally difficult. Although a few approaches have provided algorithms for exact representations of relational value functions (Boutilier et al., 2001; Hölldobler & Skvortsova, 2004; Kersting et al., 2004), these techniques have only been applied successfully to relatively simple problem descriptions. In practice, exact value function representations are difficult to obtain and the driving research question is how to efficiently find good approximations of a value function (or policy).

In this paper we attempt to address the above issues by introducing an approach to model-free relational reinforcement learning that induces structure as it learns. This approach has the advantage of allowing a learner to start with a simple relational representation and augment it as needed to learn structure that is useful for predicting the value function. We apply this learning technique to Backgammon and demonstrate that it can produce a high-performance agent with very little computational effort and in very few training games in comparison to other state-of-the-art Backgammon learning algorithms.

2. Background and Related Work

2.1. Relational Reinforcement Learning

There are two major approaches to RRL: model-based and model-free. In model-based RRL, one usually as-

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

sumes the problem is modelled *explicitly* as a Markov decision process (MDP) (Puterman, 1994) with relational state space structure (RMDP). Then it is possible to use a relational generalization of an MDP solution algorithm to solve for the value function or policy of the RMDP. In contrast, model-free RRL usually assumes an *implicit* underlying RMDP and attempts to learn the parameters (and possibly structure) of a value function via direct experience without access to the underlying model. Since we focus on model-free approaches in this work, we refer the reader to recent work this area (Sanner & Boutilier, 2005) that includes a discussion of related work.

While all of the model-free RRL approaches are too numerous to mention, many algorithms are variants of an approach that approximates the value or Q-function with logical regression trees (Dzeroski et al., 1998). While this learning approach is top-down in that it recursively partitions the state space into finer value distinctions, more recent work (Walker et al., 2004; Croonenborghs et al., 2004) has taken a bottom-up approach to finding useful features for predicting value and combining them to make a predictive estimation of state value. While our SVRRL algorithm takes this latter approach, it focuses specifically on a relational naive Bayes net representation of the value function and presents efficient techniques for learning both the value (parameters) and structure of this network under a minimum description length (MDL) framework.

2.2. Bayes Net Structure Learning

Since our goal is to learn relational naive Bayes net structure, the most relevant work along these lines is in the field of Bayes net structure learning. Friedman and Goldschmidt (1996) evaluate the tree-augmented naive Bayes (TAN) network structure for learning classifiers in propositional domains. While we opt for a naive relational Bayes network rather than a propositional TAN network, we do leverage a similar theoretical framework in our approach. In the area of learning structure in relational Bayes nets, Friedman *et al* (1999) propose techniques for learning a general class of probabilistic relational models (PRMs) from a fixed dataset. While this explicit search-based approach seems too computationally expensive to perform on-line in an RRL agent, future extensions of SVRRL could incorporate some of these ideas.

3. Relational Reinforcement Learning Framework

In this paper, we restrict ourselves to undiscounted, finite-horizon domains with a single terminal reward

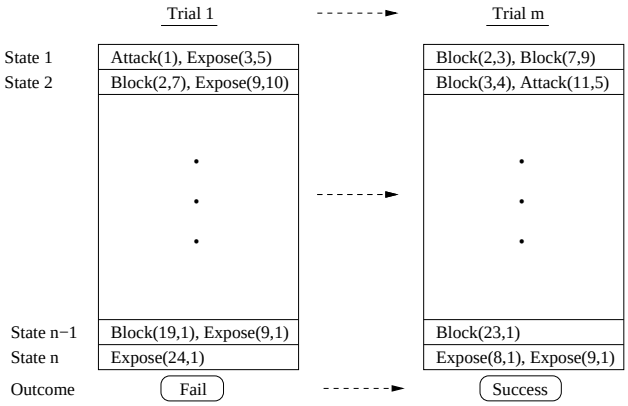


Figure 1. A diagram depicting training of the RRL agent in the domain of Backgammon (see Figure 2 for a detailed explanation of the relations). During each trial, the agent keeps track of all positive (i.e. true) ground atoms of these templates at every intermediate state (we've reduced the arity of the relations for readability). Once a terminal win or loss has been reached, the prior $P(W)$ and the conditional probability tables for the relational features $P(F_i|W)$ appearing during the trial are updated. This progresses for m trials.

of success or failure. Additionally, we assume that the state is described using a fixed set of relations $R = \{R_1, \ldots, R_i\}$, each having some finite arity. Each relation argument is assigned an attribute type from a set $A = \{A_1, \ldots, A_j\}$ where each attribute is itself a set of legal values the attribute can take. We refer to a relation and the attribute specification for its arguments as a *relation template*.

To make this more concrete, we provide a simple example domain with a single relation template $R_1(A_1, A_2)$ where $A_1 = \{a, b\}$ and $A_2 = \{1, 2\}$ so that there are four possible ground instantiations of this template: $R(a, 1)$, $R(a, 2)$, $R(b, 1)$, and $R(b, 2)$. Treating each ground atom as a binary proposition, the state is given by the full truth assignment (i.e. true or false) to each of the four propositions. Thus, in this simple relational domain, there are 2^4 or 16 possible states.

In the following discussion, we use binary propositions F_i to denote generic ground atoms (a.k.a. features) which can take on the value *true* denoted by f_i and the value *false* denoted by $\bar{f}_i$. For representational efficiency, we assume the state is represented by only the positive (true) atoms, which we arbitrarily label $\{f_1, \ldots, f_p\}$. Then, given that that there are a total of n ground atoms in a problem domain, we use absence-as-negation to infer that the remaining atoms $\{\bar{f}_{p+1}, \ldots, \bar{f}_n\}$ are false. We let $F = \{F_1 \ldots F_n\}$ (all ground atoms) and represent a state instantiation $f \in F$ as $f = \{f_1, \ldots, f_p \bar{f}_{p+1}, \ldots, \bar{f}_n\}$ (a truth assignment to all ground atoms). The omission of negative

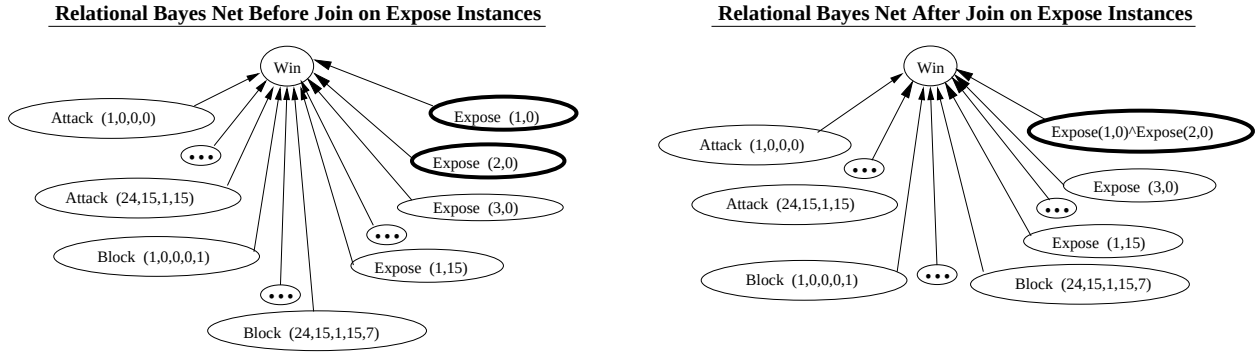


Figure 2. The relational naive Bayes net representation of the value function in Backgammon. In this domain, there are 24 points on a Backgammon board where a player's pieces can be placed (each player is assigned 15 pieces in the beginning and this number decreases as they manage to successfully bear each piece off the board). There is also a *bar* position where pieces can be placed when they have been blotted (i.e. *attacked* by the opponent because they were *exposed* by themselves on a point) and must wait to reenter the board. We use five attribute types, $PT = \{1, \ldots, 24\}$ for point locations, $OP = \{1, \ldots, 15\}$ for a count of opponents ahead of a point, $ON = \{1, \ldots, 15\}$ for a count of opponents within 7 points, $OB = \{1, \ldots, 15\}$ for a count of opponents on the bar, and $SZ = \{1, \ldots, 15\}$ as the number of consecutive points with at least two of a player's pieces (a *block* in Backgammon). From this, we define three relation templates $Attack(PT, OP, OB, ON)$, $Expose(PT, OP, OB, ON)$, and $Block(PT, OP, OB, ON, SZ)$. Here we have a child node for each ground atom derived from these templates. As SVRRL progresses, the system keeps track of the prior over winning $P(W)$ and the conditional probability tables $P(F_i|W)$ for each ground child node. As the system learns, it may decide to join two ground features as is done above for two ground atoms of the *Expose* relation.

atoms from the state is efficient since we expect the number of positive atoms to be small (and easily identifiable) in comparison to the total number of atoms ($p \ll n$). And as we will show, it is also computationally efficient for comparison of state values.

Figure 1 shows the learning task. Given a number of trials, each involving some *finite* number of time steps, the learner is presented with a relational specification of the positive state features $\{f_1, \ldots, f_p\}$ and chooses an action according to a fixed policy. This is repeated in each trial until the terminal state is reached and the terminal reward is received. If we model the underlying process as a finite-horizon MDP with a terminal reward of 1 for success/win and 0 for failure/loss, and a discount factor $\gamma = 1$ (i.e. no discount), then it is straightforward to show that the value function w.r.t. a fixed policy is simply the conditional probability of success/winning given the state, $P(w|f)$.

Now, the question we must answer is how to estimate $P(w|f)$. Even very small RRL domains can have hundreds of ground atoms and it would be impossible to represent the exact distribution, which in its fully enumerated form would require roughly one probability entry for every distinct truth assignment to ground atoms. For 100 ground atoms, this would require approximately 2^{100} distinct probability entries, which is clearly intractable. Thus, we need to focus on a compact, factored representation of $P(w|f)$ and one common way to do this is by using a Bayes net. In our case, we specifically choose to use the naive Bayes net representation given in Figure 2 since we need only record 2 probability entries $P(f_i|w)$ and $P(f_i|\bar{w})$ for each ground atom and 1 entry $P(w)$ for the prior over winning. For our previous example of a relational domain with 100 ground atoms, we need only record 201 probability entries to approximate the value of $P(w|f)$. While this is only an approximation, we show that we can "patch up" this simple representation through structure learning. But, first we focus on how to learn the value (parameters) of this network.

Now that we can compactly represent the value function as a Bayes net, we need to efficiently learn it from data. Since it is well-known that the max-likelihood parameters of Bayes net are simply the observed frequencies for each conditional probability table (CPT), we can efficiently approximate the value function by keeping track of the observed frequencies (denoted by $\hat{P}$) for each CPT. This allows us to compute the max-likelihood value for $P(w|f)$:

$$\hat{P}(w|f) = \frac{\hat{P}(f|w)\hat{P}(w)}{\hat{P}(f)} \tag{1}$$
$$= \frac{\hat{P}(w)\prod_{i=1}^{p}\hat{P}(f_i|w)\prod_{i=p+1}^{n}\hat{P}(\bar{f}_i|w)}{\sum_{o\in\{w,\bar{w}\}}\hat{P}(o)\prod_{i=1}^{p}\hat{P}(f_i|o)\prod_{i=p+1}^{n}\hat{P}(\bar{f}_i|o)}$$

As noted previously, the number of ground atoms (and therefore children in the naive Bayes network) is very

large. Yet even in the presence of an *infinite* number of negative features, we can still efficiently determine the best next state or after-state[1] given a finite set of the positive features for each state. Based on the fact that the state f which maximizes $P(w|f)$ will also maximize the log winning odds $\log(\frac{P(w|f)}{P(\bar{w}|f)})$, we obtain the following representation of the log winning odds of a state:

$$\log \frac{P(w|f)}{P(\bar{w}|f)} = \log \frac{P(w)}{P(\bar{w})} + \sum_{i=1}^{p} \log \frac{P(f_i|w)}{P(f_i|\bar{w})} + \sum_{i=p+1}^{n} \log \frac{P(\bar{f}_i|w)}{P(\bar{f}_i|\bar{w})} \quad (2)$$

Now, if we let $C = \log \frac{P(w)}{P(\bar{w})} + \sum_{i=1}^{n} \log \frac{P(\bar{f}_i|w)}{P(\bar{f}_i|\bar{w})}$, then we can express the log winning odds of a state described by *only the set of active feature instances*:

$$\log \frac{P(w|f)}{P(\bar{w}|f)} = C + \sum_{i=1}^{p} \left(\log \frac{P(f_i|w)}{P(f_i|\bar{w})} - \log \frac{P(\bar{f}_i|w)}{P(\bar{f}_i|\bar{w})} \right) \quad (3)$$

Since C is a constant *common to all states*, we can ignore it during comparisons of log winning odds of states. Thus, even in a relational naive Bayes net with a large number of negative features, it is still possible to efficiently determine the highest-valued state.

As one final practical consideration, we typically use smoothing and non-parametric techniques (e.g. nearest-neighbor, etc...) as in Sanner *et al* (2000) to efficiently store and estimate the CPTs for all ground atoms of a relation template. This allows us to leverage natural similarity measures between attribute values to obtain more robust estimates of the CPTs.

4. Structure Learning

Given the previous framework for learning the parameters of a fixed relational naive Bayes net value function in relational reinforcement learning, we now proceed to determine how to learn structure in this value function. We restrict our attention to two types of joint feature learning which we outline next. In the following examples, note that we are looking for two ground atoms F_a and F_b that we may want to join:

Feature Attribute Augmentation (FAA) When we first initialize our relational naive Bayes net for a problem domain, we obtain a child node for every ground feature, e.g. one child node may be $Expose(5, 3, 0, 2)$. Consequently, we must use the observed frequency counts to estimate the probabilities for this child node's CPT, i.e. $P(Expose(5, 3, 0, 2)|w)$ and $P(Expose(5, 3, 0, 2)|\bar{w})$. If the relation arity is high and the number of attribute choices is large, we can expect to obtain very little data for each potential ground atom of a relation template. Aside from non-parametric learning techniques for mitigating the effects of sparse data, we choose to initially approximate the above probabilities by assuming that all relation attributes are independent. For example, we estimate $P(E(5, 3, 0, 2)|w)$ (abbreviating Expose as 'E'), by $P(E(5, \cdot, \cdot, \cdot), E(\cdot, 3, \cdot, \cdot), E(\cdot, \cdot, 0, \cdot), E(\cdot, \cdot, \cdot, 2)|w)$ where $\cdot$ in an attribute slot indicates a *don't care*. This may give us a more accurate low-variance estimate in the presence of sparse data, but as we gain more experience (i.e. data) over time, we may want to relax this approximation. For example, we can let $F_a = E(5, \cdot, \cdot, \cdot)$, $F_b = E(\cdot, 3, \cdot, \cdot)$, and attempt to determine if the join $F_{a,b} = E(5, 3, \cdot, \cdot)$ is more informative than the independent features.[2]

Feature Conjunction (FC) In contrast to feature attribute augmentation, where in some sense we are simply dealing with approximations of probabilities *within the CPTs corresponding to each Bayes net child node*, we may also want to ask whether there is any additional information gained by *joining the CPTs for two arbitrary child nodes*. For example, we could let $F_a = Expose(5, 3, 0, 2)$ and $F_b = Attack(10, 3, 0, 1)$ and ask whether the joint probability of the conjunction of both atoms, $P(F_a, F_b|W)$, is more informative than the product of the probabilities given by the naive Bayes assumption, $P(F_a|W) \cdot P(F_b|W)$.[3]

In general, a learner can use both FAA and FC structure learning techniques which we denote generically as the SVRRL algorithm, or just FAA structure learn-

[1] An after-state (Sutton & Barto, 1998) is simply the state resulting from an agent's action before any other agent, if present, has chosen its respective action.

[2] For FAA-learning we are only looking at joining the attribute probability estimates in one ground atom, but it is easy to look at joining the attribute probabilities of *each* ground atom of a relation template. This latter learning approach is more relational in nature and is what we refer to by FAA-learning.

[3] For FC-learning, we are simply looking at joining two ground atoms so it would be misleading to think of this as relational learning. However, learning arbitrary conjunctions of relations for Bayes nets proves problemantic since there is no direct correspondence between a relational join and a manipulation of the underlying ground Bayes net. While relational learning of this sort has been done for Markov random fields (MRFs), we note that learning the optimal parameters for MRFs has no closed-form solution and must be done iteratively. So, for now, full FC relational learning in the naive Bayes net framework is the subject of future research. We note that full FC relational learning would also enable the use of variable unification and quantification in relational joins to reduce the number of ground atoms. Providing the RRL agent with such an expressive relational-learning space is one of our ultimate research goals.

ing which we denote as FAA-SVRRL. For whatever algorithm is chosen, the learner must maintain distributions for each individual feature $P(F_i|W)$ and potential FAA and FC joint feature instances $P(F_a, F_b|W)$. This requires a quadratic amount of work in the number of active features during max-likelihood parameter updating of the relational naive Bayes network. If this proves to be too computationally intensive for generic SVRRL, then FAA-SVRRL should be used.

Given the probabilities for our joint feature estimates, *our goal is to add relational structure to the network so that it maximizes the log-likelihood of the joint probability of the Bayes net.* Thus, given the current network structure, our goal is to ask which two feature atoms F_a and F_b to greedily combine via the FAA or FC methods of structure learning. We let V_{all} denote the set of all Bayes net binary variables $\{W, F_1, \ldots, F_p, F_{p+1}, \ldots, F_n\}$ and let $\vec{x} \in V_{all}$ be shorthand for the set of instances $\vec{x} \in \{W \times F_1 \times \cdots \times F_p \times F_{p+1} \times \cdots \times F_n\}$. Let $N(\vec{x})$ be the number of times that state instantiation $\vec{x}$ has occurred in the data. We express the log-likelihood of the naive Bayes net with parameters θ and M data samples D as the following:

$$l(\theta|D) = \sum_{\vec{x} \in V_{all}} N(\vec{x}) \Bigg(\log P(W) + \log P(F_a, F_b|W) + \sum_{i=1, i \notin \{a,b\}}^{n} \log P(F_i|W) \Bigg)$$

Now, given that we know the maximum likelihood parameters for this fixed naive Bayes network structure are simply the observed probabilities, and adding and subtracting the same $\log(\hat{P}(F_a|W)\hat{P}(F_b|W))$ term, we can express the maximum likelihood as the following:

$$\begin{aligned} l^*(\theta|D) &= M \sum_{\vec{x} \in V_{all}} \hat{P}(\vec{x}) \Bigg(\log \hat{P}(W) + \log \frac{\hat{P}(F_a, F_b|W)}{\hat{P}(F_a|W)\hat{P}(F_b|W)} + \sum_{i=1}^{n} \log \hat{P}(F_i|W) \Bigg) \\ &= M \sum_{W} \hat{P}(W) \log \hat{P}(W) + \\ &\quad M \sum_{c=1}^{n} \sum_{F_c, W} \hat{P}(F_c, W) \log \hat{P}(F_c|W) + \\ &\quad \sum_{F_a, F_b, W} \hat{P}(F_a, F_b, W) \log \frac{\hat{P}(F_a, F_b|W)}{\hat{P}(F_a|W)\hat{P}(F_b|W)} \\ &= M(H(W) + \sum_{i=1}^{n} H(F_i|W) + I(F_a, F_b|W)) \end{aligned}$$

Finally, we let $C = M(H(W) + \sum_{i=1}^{n} H(F_i|W))$ (M times the entropy of W plus the sum of the conditional entropy of every feature F_i in the network, which we note is constant *no matter what features F_a and F_b are chosen*). Thus, we arrive at the following pleasing result expressing the maximum log-likelihood of a relational naive Bayes network under a single feature join as a constant plus the the conditional mutual information values of those joined feature nodes, i.e. $l^*(\theta|D) = C + M \cdot I(F_a, F_b|W)$). If, as for FAA-learning, we want to look at the effects of joining multiple pairs of features, it is obvious that we need only sum the mutual information values of each pair being joined. Thus, FAA and FC-learning require local evaluations only, thereby leading to a highly efficient structure learning framework.

Since random noise almost always guarantees non-zero mutual information, we need a principled way to control the amount of structure learned. For this purpose, we choose the minimum description length principle (MDL) commonly used in Bayes net learning (Lam & Bacchus, 1994) to allow the SVRRL algorithm to balance the amount of training experience with the complexity of the network structure. At each update, we check whether to add a feature by determining if it minimizes the following MDL score where $|B|$ is the number of parameters in the naive relational Bayes network B:

$$MDL(B|D) = \frac{1}{2} log(M|B|) - l^*(\theta|D) \qquad (4)$$

Since this computation involves only a negation and constant addition to the log-likelihood, the update check to determine whether a joint feature should be added can be computed quite efficiently.

Finally, we can briefly summmarize the full SVRRL algorithm: 1) Initialize the naive Bayes network for a domain with a child for each ground atom, begin by estimating the probability for each child CPT by treating the individual relational attributes independently within each node (see FAA-learning); 2) For the current trial, execute the policy[4] to obtain data and keep track of all individual and joint feature occurrences; 3) When the MDL principle permits, augment the child CPTs (FAA) or child nodes (FC) with the feature joins which maximize the MDL score; 4) When a terminal state is reached, update all probability estimates (individual and joint) for features encountered during the trial, and goto step 2 to begin the next trial.

[4]The policy can be determined on-line as the best state w.r.t. the current value function. Although convergence is not guaranteed for such a non-stationary policy, this approach often works well in practice.

Player	Winning PCT	# Training Games
TD-Gammon 1-Ply (est)	66.0 % ± ???	1,500,000
FAA-SVRRL	58.6 % ± 0.02	5,000
Pubeval	50.0 % ± 0.00	unknown
HC-Gammon	40.0 % ± 3.46	100,000

Table 1. Asymptotic winning percentage of various Backgammon programs vs. Pubeval.

5. Empirical Evaluation

Since the SVRRL algorithm is intended to handle domains with only terminal rewards of success or failure, such an approach is appropriate for learning in goal-oriented tasks such as games where the outcome is simply a win or a loss. We choose Backgammon as a testbed for empirical evaluation since it has a rich relational feature space, a high branching factor, and is heavily stochastic. This makes it an extremely difficult game to solve via model-based techniques, making it a good candidate domain for putting SVRRL to the test.[5]

We used FAA-SVRRL as our initial implementation of a Backgammon learning agent. While we lack space to show the graphs, we note that FAA-SVRRL learns more quickly and asymptotically outperforms an algorithm using random structure learning in place of the greedy-optimal structure learning outlined previously. Table 1 gives the asymptotic performance and number of training games of the converged FAA-SVRRL learning algorithm vs. PubEval (trained linear neural net) in comparison to results obtained for an estimate of TD-Gammon 2.1 with 1-ply search (Tesauro, 1992) (expert level)[6], PubEval, and HC-Gammon (Pollack et al., 1996), a neural net learned via genetic coevolution. We note that SVRRL not only converges in the least number of training games, but achieves a commendable level of performance, coming in second only to expert-level TD-Gammon.

[5]In support of our efficiency claims, we note that our FAA-SVRRL learning algorithm completed 5000 training games in under 10 minutes of computation time on a 1 GHz Pentium III with 128 Mb of RAM. The best converged learner required only 240 features (or less than 10 Kb of RAM) to store the full non-parametric representation of the conditional probability tables. This is a reasonably compact representation of a value function for a game estimated to have over 10^{18} distinct states.

[6]The TD-Gammon 1-Ply value is estimated from (Galperin & Viola, 1998) assuming that the *Lin-3* opponent referenced in this paper performs comparably to Pubeval. This seems to be a reasonable assumption since both are reasonably strong players based on a single-layer linear neural net evaluator.

6. Concluding Remarks

In this paper, we have motivated and examined the problem of learning both structure and value in a relational reinforcement learning framework. This algorithm is not only extremely efficient, involving simple updates and no search, but by learning only the structure that maximizes the log likelihood of the relational naive Bayes net under a minimum description length framework, the computational burden on the learning agent is minimized. We have applied our FAA-SVRRL agent to the domain of Backgammon and have shown that it learns useful structure in an extremely efficient manner while achieving a commendable asymptotic performance level.

Acknowledgments

The author would like to thank Martijn van Otterlo and the anonymous reviewers for their comments and suggestions regarding earlier versions of this paper.

References

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. *IJCAI-2001.*

Croonenborghs, T., Ramon, J., & Bruynooghe, M. (2004). Towards informed reinforcement learning. *ICML-2004 Workshop on Relational Reinforcement Learning.*

Dzeroski, S., de Raedt, L., , & Blockeel, H. (1998). Relational reinforcement learning. *ICML-1998.*

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. *IJCAI-1999.*

Friedman, N., & Goldszmidt, M. (1996). Building classifiers using bayesian networks. *AAAI-1996.*

Galperin, G., & Viola, P. (1998). *Machine learning for prediction and control* (Technical Report). MIT.

Hölldobler, S., & Skvortsova, O. (2004). A logic-based approach to dynamic programming. *AAAI-2004 Workshop on Learning and Planning in Markov Processes.*

Kersting, K., van Otterlo, M., & de Raedt, L. (2004). Bellman goes relational. *ICML -2004.*

Lam, W., & Bacchus, F. (1994). Learning bayesian belief networks: An approach based on the mdl principle. *Computational Intelligence*, *10*, 269–294.

Pollack, J., Blair, A., & Land, M. (1996). Coevolution of a backgammon player. *Fifth Artificial Life Conference.* Nara, Japan.

Puterman, M. (1994). *Markov Decision Processes-Discrete Stochastic Dynamic Programming.* New York, NY: John Wiley and Sons, Inc.

Sanner, S., Anderson, J. R., Lebiere, C., & Lovett, M. (2000). Achieving efficient and cognitively plausible learning in backgammon. *ICML-2000.*

Sanner, S., & Boutilier, C. (2005). Approximate linear programming for first-order mdps. *UAI-2005.*

Sutton, R. S., & Barto, A. (1998). *Reinforcement learning: An introduction.* MIT Press.

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinforcement learning: An overview. *ICML-2004 Workshop on Relational Reinforcement Learning.*

Tesauro, G. (1992). Practical issues in temporal difference learning. *NIPS-92.*

van Otterlo, M., & Kersting, K. (2004). Challenges for relational reinforcement learning. *ICML-04 Workshop on Relational Reinforcement Learning.*

Walker, T., Shavlik, J., & Maclin, R. (2004). Relational reinforcement learning via sampling the space of first-order conjunctive features. *ICML-04 Workshop on Relational Reinforcement Learning.*

Small World Network Based World Representation for Scalable Reinforcement Learning

Seung-Joon Yi SJLEE@BI.SNU.AC.KR
Byoung-Tak Zhang BTZHANG@BI.SNU.AC.KR
School of Computer Science and Engineering, Seoul National University, Seoul 151-742, South Korea

Abstract

The curse of dimensionality plagues practical uses of reinforcement learning. Temporal abstraction approaches have been proposed to overcome this problem, but typically they require a priori design of the hierarchy and lack the compact representation needed for large sized problems, so their practical uses are still limited. Inspired by recent research in complex networks, we present a compact self-organizing, growing network for world representation to scale up reinforcement learning. Continuous state space is represented with a compact self-organizing network, and the network is augmented to have small world property without a priori knowledge. Experimental results with various problem sizes show that the average path length between nodes of this network scales subpolynomially with the size of the network, and the convergence of reinforcement learning is accelerated significantly.

1. Introduction

In the framework of Reinforcement Learning (RL), an agent attempts to learn a policy, i.e. a mapping from state to action, that maximizes some time aggregate of rewards. In the traditional RL framework, the environment is defined as a discrete-time, discrete-state Markov decision process (MDP). Popular RL algorithms such as Q-Learning assume tabular representation of both the state and the action space, and estimates the values of all the state-action pairs to find the optimal policy (Watkins & Dayan, 1992). However, in most real world problems with continuous or high-dimensional state spaces, it is impossible to enumerate all of the state-action pairs. Even with the problems of discrete state space, it is impractical to estimate the value function for all the states when the problem size is large. So it is necessary to use some kind of compact world representation schemes.

A common solution to large or continuous state spaces is using a function approximator such as neural networks (Boyan & Moore, 1995). Using a function approximator with RL has shown good results in some situations. However, it is also known that the number of parameters to be estimated grows exponentially with the size of any compact encoding of a state (Barto & Mahadevan, 2003). Attempts to combat this curse of dimensionality lead to temporal abstraction where decisions are not required to perform every single action. This naturally leads to hierarchical control architectures and thus the associated learning algorithms are called hierarchical reinforcement learning (HRL) algorithms. However it is still hard to use these approaches directly to real world tasks. The first problem is that the structure of hierarchy, subgoals, sub MDPs and subtasks should be decided in advance. The user should 'program' with the problem specific knowledge. Another problem is that these hierarchical RL approaches still assume a tabular representation of states and actions which makes it difficult to apply the HRL algorithms directly to large sized problems that really need them.

Meanwhile, recent studies of complex networks show that many real-world networks, such as web graphs and social networks, show the small world property where most pairs of nodes are linked by short chains of nodes (Watts & Strogatz, 1998). Based on this fact, we propose a new approach of building temporal abstraction using the small world network models. The MDP is augmented with subtasks whose structure is determined by the corresponding small world network models, where two states in the augmented MDP need small number of decision steps between them.

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

And for the practical application to the continuous state real world problems, we also utilize an incremental network that adaptively maps sensory input to actions. By augmenting the network to have the small world property and using appropriate navigation algorithms to select actions, we can build a scalable RL algorithm with compact representation without problem specific knowledge.

This paper is organized as follows. In Section 2, we briefly review the RL framework and the properties of complex networks. In Section 3, we present a practical reinforcement learning algorithm with small world network representation of the environment. In Section 4, we describe the experimental setup and the results. Finally, in Section 5, we conclude with a few future directions.

2. Related Works

2.1. Reinforcement Learning and Information Propagation on Networks

Here we briefly review the standard reinforcement framework of discrete time, finite MDPs. In this framework, a learning agent interacts with an environment at discrete time scale. On each time t, the agent chooses an action $a_t \in A$ using its policy based on the state $s_t \in S$ it perceives. Then the environment gives the agent a numerical reward $r_{t+1} \in R$ and moves to the next state s_{t+1}. The objective of the agent is to learn a policy, mapping from states to actions, that maximizes the expected discounted future reward defined as

$$R_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \quad (1)$$

To learn the optimal policy, we can get the optimal action value function $Q^*(s,a)$ using the following Q-learning method (Watkins & Dayan, 1992)

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)]. \quad (2)$$

We can generalize the MDP framework to Semi-MDP framework where each action a can take variable amounts of time $k(s,a)$ (Sutton et al., 1999). A slightly different update rule can be used such as

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma^{k(s,a)} \max_{a'} Q(s',a') - Q(s,a)]. \quad (3)$$

In the update rules (2), (3), action value functions of a state s, $Q(s,a)$ are updated using the value functions of its successor state s', $Q(s',a')$. This can be viewed as the propagation of information from state s to its successor state s'. When the agent receives positive reward in a certain state, the information of that reward is propagated to adjacent states, in the form of action value function. When each state collects enough amount of information, their action value function converges and the RL problem is solved. By making the propagation of information faster by adopting temporal abstraction, we can expect better efficiency in solving RL problem.

2.2. Complex Networks

Recent researches on complex networks have showed that most of the real world networks share the following three features. (a) Small world property. There exists a short path between any two nodes, compared to their size (Watts & Strogatz, 1998). (b) High clustering coefficient. Two nodes with a common neighbor has much more likely to be connected than two nodes without one. (c) Scale free degree distribution. The distribution of the degree decays as a power law, which is invariant to scaling. This property is often related to the hierarchical organization of the network.

Various network models with these properties are suggested. Here we present two small network models we will use for our task. First model is the Kleinberg's model (Kleinberg, 2001). In this model, we start with a regular lattice network and random long links (u,w) are added to current network with probability proportional to $d^{-\alpha}$ where d is the lattice distance from u to w. It is known that if we set the value of α to the dimension of underlying lattice, a decentralized greedy algorithm can achieve polylogarthmic search time. Second model is the scale free growing network model (Barabási & Albert, 1999). In this model, when a new node u is added to current network random long links (u,w) are added with probability proportional to degree of w, which is called the linear preferential attachment. This model also shows the scale free degree distribution.

3. Building a Small World Network for World Representation

Temporal abstraction approaches reduce the number of decision steps between states by augmenting the MDP with subtasks. But they require a priori knowledge of the problem in most cases. Instead of using the problem specific knowledge, we propose to use the network model with small world property to determine the structure of hierarchy. By the small world property, the augmented MDP will have small number of decision steps between states, which will help solving

RL problem efficiently.

3.1. The Small World Network-Based World Representation

To apply RL for continuous problem, we have to approximate the state space by discretization or using function approximators such as neural networks. To adopt two small world networks we mentioned above, we need an incremental network model with a regular lattice structure. An example of such a model is the growing neural gas network(Fritzke, 1995). Similar approaches were used in(Gross et al., 1998; Toussaint, 2003) for world representation. Its online extension is the incremental topology preserving map(Millan et al., 2002) which we will use as the base of our algorithm.

Our algorithm is summarized in figure 1. It incrementally adds nodes to the unexplored regions of state space, and uses self-organization rule to modify the connectivities and positions of nodes. Links between edges are modified so that each node is only connected to its neighbors, and the position of nearest node and the positions of all its neighboring nodes are moved closer to the input position. Furthermore, long range edges are added to make the network have the small world property. If we assume that all the state transitions are local, we can use this network model as the discrete MDP which approximates the original problem. Each node represents a corresponding region of state space, and each edge correspond to a subtask of moving to a specific state in original problem. This approach has much in common with topological mapping approaches in robotics (Thrun, 1998) and node graph approaches used in many 3D games (Rabin, 2002).

Our approach has two major differences from other network based approaches. The first difference is the existence of long links. When a new node is added, a long link denoting the corresponding subtask is added with the probability given by a small world network model. This procedure makes the resulting network to have the small-world property. The second difference is the constraint of visibility, which requires that there should be a straight, non-blocked path between two linked nodes in state space. If this requirement is met, the optimal subpolicy for a subtask is trivially given as moving straightly to its subgoal.

3.2. Reinforcement Learning in a Small World Network

Now we need an appropriate RL algorithm for the small world network based world model. To select actions, the easiest way is using conventional action selection methods such as ϵ-greedy method. Although

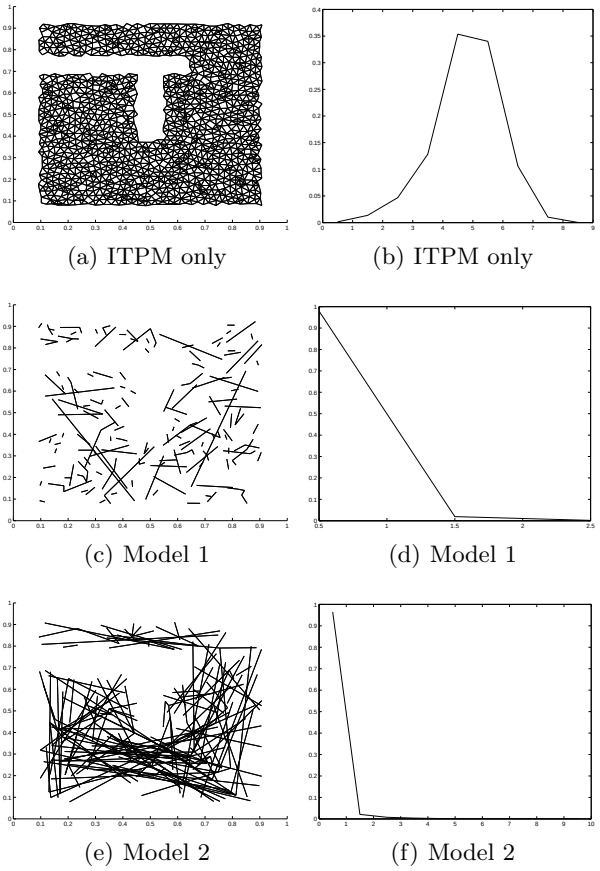
1. Perceive the current position x.
2. Find the nearest node b visible from x
   and second nearest node b' visible from x.
3. If the distance between x and b exceeds the unit radius r, then
   (A) Add a node u at x.
   (B) Create edges from u to b and b',
   (C) Remove any edge between b and b',
   (D) Select nodes v visible from x according to probability proportional to:
   (MODEL 1) $dist(u,v)^{-\alpha}$, where $dist(u,v)$ is the euclidian distance between u and v.
   (MODEL 2) $d(v)$, the number of long range edges starting from v.
   (E) Create long range edges from u to v with the probability p_l.
   Else
   (F) Create an edge between b and b' if b is visible from the position of b'.
4. Move w_b, the position of b, and w_r, the positions of all its neighboring nodes r, toward x if possible:
   $w_b \leftarrow w_b + \delta(x - w_b)$
   $w_r \leftarrow w_r + \delta_r(x' - w_r)$

Figure 1. An algorithm to build a small world network based world model.

this method ignores the network structure, augmenting MDPs with temporal abstraction alone can help reinforcement learning process (Sutton et al., 1999). And it is also known that random walking in a scale free network gravitate towards the high degree nodes, making the search more efficient (Adamic et al., 2001). So for the simplicity, we use a simple ϵ-greedy action selection rule in this work. To update the value functions, we can directly use Semi-MDP value update rule (3) for both of the models we described above. Finally, we need to get a subpolicy for each subtask. For the navigation task we are interested in, finding subpolicies can be trivial. However, for a general RL task, we have to use a local policy learning algorithm such as the Experience Replay procedure(Lin, 1992). We leave this as a future work.

4. Experimental Results

In this section we describe two experiments. For both experiments, we first let the agent explore the state space and use our algorithm to generate Semi-MDP network. And we run standard Q-learning algorithm on the network. Two versions of proposed algorithm, each using model 1 and model 2 we discussed above, are tested against standard ITPM algorithm. Common parameter values we use are as follows: movement parameter δ=0.0002, δ_r=0.00002, long link ratio



(a) ITPM only (b) ITPM only

(c) Model 1 (d) Model 1

(e) Model 2 (f) Model 2

Figure 2. Example of networks generated using each algorithms and their degree distributions. Networks are generated with r=0.02 and degree distributions are measured with r=0.00625. Only long links are shown for Models 1 and 2.

p_l=0.1. The lattice dimension p is empirically determined from the average degree of the baseline network. Various values for unit radius r are used to examine the scaling behaviors of each algorithms.

Our experiment is greatly simplified by the visibility constraint, by which the optimal subpolicy for a subtask is trivially given as going straight to their subgoal position. Although this approach is not suitable for general RL tasks where visibility constraint is not applicable, it suits well for navigation task we are interested in. For reinforcement learning part, we use simple epsilon-greedy action selection rule with ϵ=0.1, α=0.5, γ=0.9. we used the Semi-MDP update rule (3), using the length of edges normalized by the unit radius r as the execution time $k(s, a)$. We will discuss each experiments further in following subsections.

4.1. 2D Puddleworld

The first experiment uses a [0,1) by [0,1) continuous state space with a T-shaped obstacle in it. At each

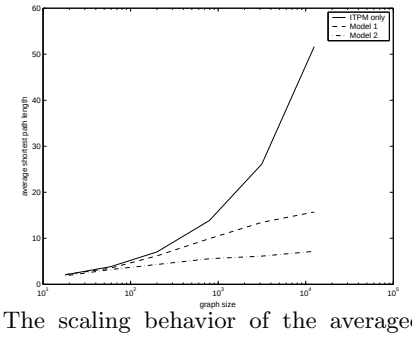
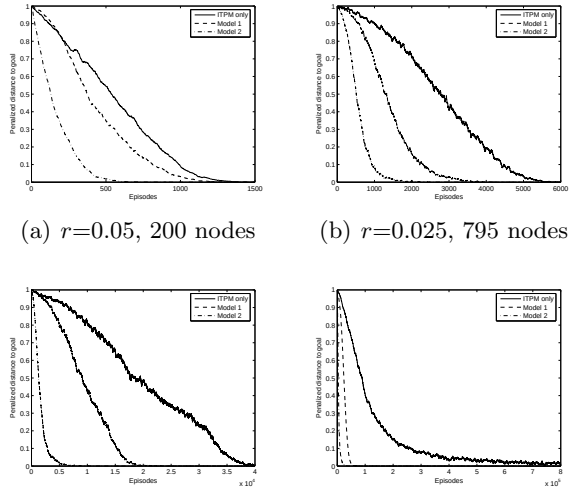


Figure 3. The scaling behavior of the averaged shortest path length. In contrast to baseline ITPM which shows a polynomial growth of shortest length, model 1 and model 2 show a polylogarithimic growth of shortest path length which is shown as the straight line on semi-log plot.

time step the agent can move to any direction, with step size $r/10$. We let the agent do random walk and generate networks for each models using unit radius r=0.2,0.1,0.05,0.025,0.02,0.0125,0.01 and 0.00625. Generated networks from each models and their degree distributions are shown in figure 2. Figure 2a shows the baseline network ITPM generates where each node is connected to only its neighbors. From figure 2b, we can see the degree distribution of the network is peaked at 5, and the lattice dimension is determined as p=$log_2 5$=2.322. Figure 2b and 2c show the additional long links model 1 and 2 add to the baseline network. From figure 2f, we can see the emergence of hub structures and the characteristic power law curve of degree distribution in model 2. The average shortest path length of each network is shown in figure 3. We can see that in contrast to the baseline network where the average shortest path length grows linearly with problem size, the average shortest path length grows polylogarithmically in model 1 and model 2.

Finally we run a Q-learning algorithm on generated networks. At each episode, the agent starts at a random start point and move up to 100 steps. In an absorbing goal area positioned upper right corner of the state space, a reward of 100 is given. The number of episodes in a run is empirically determined to fully cover its convergence phase. The total distance from start point to goal is measured at each episode. To penalize the episodes that fail to reach the goal within 100 unit distance, we assign 1000 unit distance as the penalized distance to the goal of that episode. We run a number of runs and average the penalized distances to goal. Figure 4 shows the convergence of penalized distance to goal for various network sizes. We can see that using model 1 and 2 improves the convergence speed of RL, and this effect is more apparent in bigger problem. To see the scaling behavior of RL performance over problem size, we measure the number of episodes needed to reach 50% convergence from each



(a) r=0.05, 200 nodes (b) r=0.025, 795 nodes

(c) r=0.0125, 3157 nodes (d) r=0.00625, 12501 nodes

Figure 4. The penalized distances to the goal at each episode, averaged after 100 runs. 1000 unit distance is used to penalize the failed episodes. Each graph is normalized to fit in the range of 0 to 1.

learning curves. To reduce the effect of variance, we apply a smoothing filter which averages outcomes within a given window size. Figure 5 shows the scaling behaviors of RL using each models. Though they do not scale polylogarithmically like shortest path length, we can accelerate the convergence of reinforcement learning algorithm significantly.

4.2. 3D Gameworld

To show that our algorithm can handle a complex problem with a continuous state space, we now consider a 3D gameworld task. For our experiment platform, we use a popular 3D game Half-Life 2 (Hodgson, 2004) which enables the real-time simulation of continuous 3D world. We use a map named dm_lockdown whose size is approximately 50*150*10 m. Learning agent has size 1*1*1 m and has maximum movement speed of 7 m/s (27k mph). The position of agent is checked every 1/20 second to update the network and get a new action. To explore the state space efficiently, we use a RL based multi agent exploration algorithm, which we do not cover in this paper.

We generated three networks using unit radius r=0.5,0.75,1.0 m. Generated networks consist of 4594, 6403, 13252 nodes and 6933, 11447, 13866, 23341 links, respectively. Long links generated by model 1 and 2 account for about 5% of total links. Generated network using r=0.75 meter is shown in Figure 6. With the same setting we use for the first experiment, we run Q-learning on each networks. A reward of 100 is given at the single goal located in one of the rooms. Figure 7

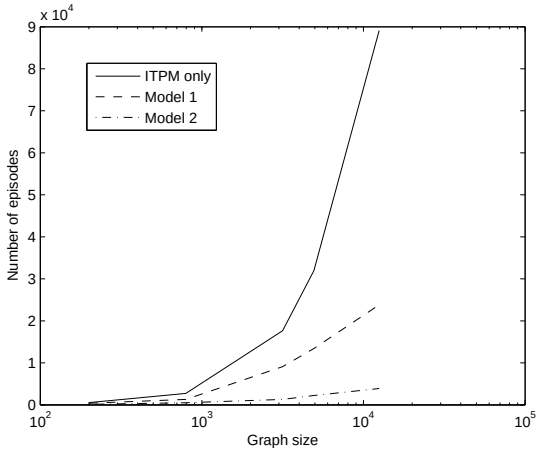


Figure 5. The scaling behavior of the number of episodes to reach 50% convergence.

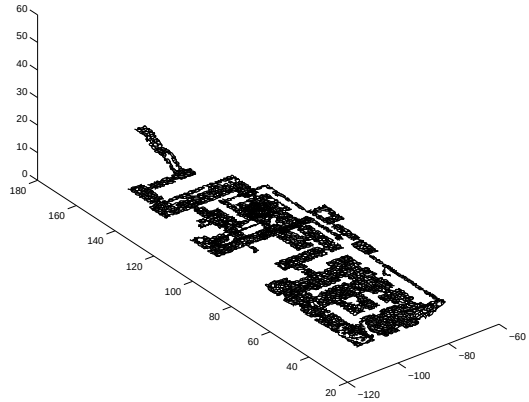
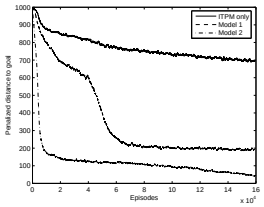


Figure 6. The 3D view of generated baseline network using r=0.75 m. Total number of nodes is 6403.

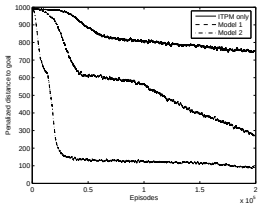
shows the convergence properties of penalized length to goal for each models. In contrast to the baseline ITPM algorithm which shows very slow convergence speed, our algorithm quickly start to converge.

5. Conclusion and Future Work

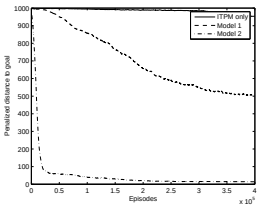
We propose a novel network based world representation to cope with the curse of dimensionality in reinforcement learning. By augmenting the network with a small number of additional links based on small world network models, we demonstrate that we can keep the growth of the number of decision steps polylogarithmically, which can accelerate the convergence of RL as the problem size grows bigger. Experimental result with 3D game environment shows that our algorithm can learn usable policy in reasonably short time, even with a huge problem with state size exceeding 10,000. Although still preliminary, our approach is promising



(a) r=1, 4594 nodes



(b) r=0.75, 6403 nodes



(c) r=0.5, 13252 nodes

Figure 7. The penalized distances to goal at each episode for a 3D gameworld task, averaged after 10 runs. The 1000 unit distance is used as the penalized distance for failed episodes.

in several aspects: it is scalable, takes continuous state and action space, has a compact representation and does not need problem specific knowledge.

There are many interesting directions for future work. The most interesting one is extending our approach to general continuous state space RL problems by adopting a subpolicy learning algorithm. Using larger subtasks can help reducing the number of decision steps between states, but it may add overhead of learning subpolicies for larger subspaces. Finding the right tradeoff between these two will be a challenging problem.

References

Adamic, L. A., Lukose, R. M., Puniyani, A. R., & Huberman, B. A. (2001). Search in power-law networks. *Phys. Rev. E*, *64*, 46135–46143.

Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, *286*, 509–512.

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Systems journal*, *13*, 41–77.

Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems 7* (pp. 369–376). Cambridge, MA: The MIT Press.

Fritzke, B. (1995). A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky and T. K. Leen (Eds.), *Advances in neural information processing systems 7*. Cambridge MA: MIT Press.

Gross, H., Stephan, V., & Krabbes, M. (1998). A neural field approach to topological reinforcement learning in continuous action spaces. *IEEE World Congress on Computational Intelligence, WCCI'98 and International Joint Conference on Neural Networks, IJCNN'98.*

Hodgson, D. (2004). *Half-life 2:raising the bar*. Prima Games.

Kleinberg, J. (2001). Small-world phenomena and the dynamics of information. *Advances in Neural Information Processing Systems 14*. Cambridge MA: The MIT Press.

Lin, L. G. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*, 293–321.

Millan, J. D. R., Posenato, D., & Dedieu, E. (2002). Continuous-action q-learning. *Machine Learning*, *49*, 241–265.

Rabin, S. (2002). *Ai game programming wisdom*. Charles River Media, Inc.

Sutton, R. S., Precup, D., & Singh, S. P. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial intelligence*, *99(1)*, 21–71.

Toussaint, M. (2003). Learning a world model and planning with a self-organizing, dynamic neural system. *Advances in Neural Information Processing Systems 16*. Cambridge, MA: The MIT Press.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, *393*, 404–407.

Multigrid Algorithms for Temporal Difference Reinforcement Learning

Omer Ziv
Department of Electrical Engineering, Technion - Israel Institute of Technology, Haifa 32000, Israel

Nahum Shimkin SHIMKIN@EE.TECHNION.AC.IL
Department of Electrical Engineering, Technion - Israel Institute of Technology, Haifa 32000, Israel

Abstract

We introduce a class of Multigrid based temporal difference algorithms for reinforcement learning with linear function approximation. The proposed Multigrid-enhanced TD(λ) algorithms allows to accelerate the convergence of the basic TD(λ) algorithm while keeping essentially the same per-sample computational cost. The convergence properties of the algorithm are discussed along with an illustrative simulation example.

1. Introduction

Temporal difference algorithms for evaluating the value function of a given policy are a central component in many Reinforcement Learning (RL) schemes. The basic TD(λ) algorithm was introduced in Sutton (1988). In this paper we propose a Multigrid-based enhancement of the TD(λ) algorithm, which aims to improve the convergence rate while retaining the same $O(K)$ complexity per iteration, where K is the number of parameters to be learned.

Multigrid (Briggs, 2000; Trottenberg, 2001) is a well-established approach to accelerate iterative solutions of large sets of linear equations, such as those arising in the numerical solution of partial differential equations. Essentially, an iterative relaxation scheme at a fine resolution level is augmented by a coarse-grid correction which reduces the so-called "smooth" error components, which are otherwise slow to converge. Applying this correction recursively over several resolution levels leads to a Multigrid scheme. When applied to value iteration or TD(λ) with linear function approximation, this approach leads to algorithms that operate with different sets of basis functions, each intended to capture a different resolution level of the problem. We shall focus in particular on the Algebraic Multigrid (AMG) variant of Multigrid, which allows the automatic construction of the coarse grid hierarchies based on the system matrices. This opens up interesting possibilities for the automatic construction of basis function hierarchies.

The proposed Multigrid schemes are closely related to other multi-scale methods of Dynamic Programming and to hierarchical approaches to learning; pointers to this literature may be found in (Boutillier, Dean, & Hanks, 1999; Barto & Mahadevan, 2003).

The paper is structured as follows. Sections 2 and 3 provide the necessary background on Multigrid and TD(λ), respectively. Section 4 outlines the (straightforward) application of Multigrid to value iteration for policy evaluation in a non-learning scenario. Section 5 presents the Multigrid learning algorithms are their analysis, while Section 6 presents an illustrative simulation experiment. We note that many details have been omitted in the present version of the paper due to space limitations. Full details may be found in (Ziv, 2004).

2. Multigrid Basics

We consider the efficient solution of the system of linear equations $Ax = b$, where A is a square matrix and typically sparse. Standard iterative methods are of the form $x := x + Q^{-1}(b - Ax)$, where Q stands for a scaled identity matrix (Richardson iteration), the diagonal of A (Jacoby relaxation), or its lower-triangular part (Gauss-Seidel). When the *smoothing matrix* $(1 - Q^{-1}A)$ has eigenvalues close to the unit circle, the corresponding error components are slow to converge. Such error components are referred to

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

as "smooth", and typically correspond to "low frequency" components in a geometric context. Multigrid uses coarse-level corrections to reduce these smooth error components.

A multigrid structure comprises of: (a) A sequence of subsequent resolution levels indexed by $\ell \in \{0, 1, \ldots, \ell_{\max}\}$, with $\ell = 0$ the finest; (b) A corresponding set of equations $A_\ell x_\ell = b_\ell$ of dimension n_ℓ, where A_0, b_0 are the primary (fine-resolution) system matrices, A_ℓ, b_ℓ represent the system equations at resolution level ℓ , and $n_{\ell+1}$ is several times smaller than n_ℓ (a factor of 4 is common for 2D problems); (c) *Restrictor operators* $I_\ell^{\ell+1}$ which turn a solution x_ℓ into an approximate solution $x_{\ell+1} = I_\ell^{\ell+1} x_\ell$ of the next-coarser level; and (d) *Interpolators* $I_{\ell+1}^\ell$ ($n_\ell \times n_{\ell+1}$ matrices), which do the opposite.

A basic two-level coarse grid correction at level $\ell < \ell_{\max}$ proceeds as follows. Starting with an initial candidate x_ℓ, an approximate solution to the equation $A_\ell x_\ell = r_\ell$ (with r_ℓ to be defined shortly) is obtained as follows:

1. Presmoothing: Apply a (small) number of iterative relaxations $x_\ell := x_\ell + Q_\ell^{-1}(r_\ell - A_\ell x_\ell)$
2. Compute the residual $\text{res}_\ell = r_\ell - A_\ell x_\ell$, and restrict to the next level: $r_{\ell+1} = I_\ell^{\ell+1}\text{res}_\ell$
3. Approximately solve $A_{\ell+1} x_{\ell+1} = r_{\ell+1}$
4. Apply correction: $x_\ell := x_\ell + I_{\ell+1}^\ell x_{\ell+1}$
5. Postsmoothing: Similar to presmoothing.

By recursively applying this procedure at step 3 we obtain a multi-grid scheme. A standard *V-cycle* starts at level 0 with $r_0 = b_0$ and proceeds all the way down to level $\ell_{\max}$ and back up. Note that the system vectors b_ℓ (for $\ell \geq 1$) do not play any role here as they are replaced by the interpolated residuals r_ℓ. At the coarsest level $\ell = \ell_{\max}$ the dimension is typically chosen to be sufficiently small so that the equation $A_\ell x_\ell = r_\ell$ may be solved exactly. The whole scheme is usually initialized with some "coarse to fine" procedure which does utilize the system vectors (b_ℓ).

In classical (geometric) Multigrid, the system equations at the different levels are typically obtained by discretizing the original (continuous) problem over a regular grid at different resolutions. The inter-level (restriction and interpolation) operators are then constructed.

2.1. Algebraic Multigrid (AMG)

AMG (Brandt, McCormick & Ruge, 1984; Stüben, 2001) takes a different approach. Here the multigrid structure is constructed automatically in a setup phase from the initial system matrices A_0, b_0, based only on the algebraic structure of A_0, and without any "higher level" information on the problem. This makes AMG attractive as a "black box" solver for sparse linear equations, whether of geometric origin or not.

The AMG setup phase proceeds recursively, starting at $\ell = 0$. First the inter-level operators $I_{\ell+1}^\ell$ and $I_\ell^{\ell+1}$ are constructed based on A_ℓ. The system matrices for the next level are then defined, typically via the *Galerkin operator* $A_{\ell+1} = I_\ell^{\ell+1} A_\ell I_{\ell+1}^\ell$ (and $b_{\ell+1} = I_\ell^{\ell+1} b_\ell$ if required). This proceeds until the dimension of $\mathbf{A}_\ell$ is sufficiently small for a direct solution.

Several procedures exist for setting up the inter-level operators. The guiding principle is to allow any *algebraically smooth* error vector to be well approximated over the next level, namely by some interpolated vector of that level. The scheme used in our simulations is the Ruge-Stüben algorithm with direct interpolation (Stüben, 2001; section A.7). We have also used for comparison a related *state aggregation* scheme, where each fine-level variable s interpolated from a single coarse-level variable. For details see (Ziv, 2004).

Multigrid *theory* aims to establish convergence of the iterative algorithm and, more importantly, to provide bounds on the convergence rate and guidelines for algorithm improvement. A well developed theory currently exists mainly for problems in which the system matrix A is symmetric and positive-definite (s.p.d.), and, in particular, when A is also an M-matrix (namely s.p.d. with negative off-diagonal elements) and diagonally dominant. In practice, properly planned algorithms (and AMG in particular) are robust with respect to violation of these assumptions.

3. MDPs and the TD(λ) Algorithm

Consider a Markov Decision Process (MDP) with state $\mathcal{S}$ and action space $\mathcal{A}$. We assume here a finite state space (but note that the proposed learning algorithms are applicable to more general state spaces due to the use of basis functions). Given the state s_t and action a_t at time t, a reward $g_t = g(s_t, a_t)$ is obtained, and the next state s_{t+1} is determined according to the stationary transition probability $p(s_{i+1}|s_t, a_t)$. A *stationary policy* π is a mapping $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $\pi(s, a)$ is the probability of taking action a at state s. Fixing the policy π, the state process becomes a Markov chain with transition probabilities $p(s'|s) = \sum_a \pi(s, a) p(s'|s, a)$, and expected rewards $g(s) = \sum_a \pi(s, a) g(s, a)$. We shall assume that the induced Markov chain is *irreducible*, *a-periodic*, with a

unique stationary distribution $q(s)$. For future reference we denote by P the transition matrix with $P_{s,s'} = p(s'|s)$, the reward vector g with elements $g_s = g(s)$, and the diagonal matrix D with $D_{ss} = q(s)$. We consider the discounted cost functional with a discount factor $\gamma \in (0,1)$, namely $v(s) = E(\sum_{t=0}^{\infty} \gamma^t g_t | s_0 = s)$. The function $v(s)$ of the stationary policy π is well known to be the unique solution of the Bellman equation

$$(I - \gamma^P)v = g \tag{1}$$

where I denotes the identity matrix and v is a vector of state values, i.e. $v_s = v(s)$. The value function is approximated as a linear combination of K basis functions $\phi^k : S \to \Re, k = 1, \ldots, K$, namely

$$v(s) \approx \sum_{k=1}^{K} \phi^k(s)\theta^k = \phi(s)^T\theta$$

where $\phi = (\phi', \ldots, \phi^K)$, and $\theta \in \Re^K$ is the parameter vector to be tuned. The TD(λ) algorithm iteratively applies the following update rule

$$\theta_t = \theta_{t-1} + \alpha_t z_t \Big(g_t - (\phi(s_1) - \gamma\phi(s_{t+1}))^T \theta_{t-1} \Big);$$
$$z_t = \lambda\gamma z_{t-1} + \phi(s_t)$$

where $z_t = (z_t(s), s \in S)$ is the *eligibility trace* vector, initialized by $z_0 = 0$. $\lambda \in [0,1]$ is the algorithm parameter, and α_t is a positive *gain* sequence.

Theorem 1 (Tsitsiklis & Van Roy, 1997) *Assume that: (i) $\sum_{t=0}^{\infty} \alpha_t = \infty$, $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$. (ii) The basis functions are linearly independent. Then TD(λ) converges (w. p. 1) to the unique solution θ^* of*

$$A\theta^* = b \tag{2}$$

where

$$\begin{aligned} A &= \Phi^T(I - \gamma\lambda P)^{-1}D(I - \gamma P)\Phi \\ b &= \Phi^T D(I - \gamma\lambda P)^{-1}g \end{aligned} \tag{3}$$

and Φ is the $N \times K$ matrix with basis functions as its columns, namely $\Phi_{sk} = \phi_k(s)$.

4. AMG for Value Iteration

In this section we briefly consider value iteration for the known model case. Here the application of AMG as a "black box" solver to (1) is straightforward, by defining $A = I - \gamma P$ and $b = g$. Observe that standard value iteration, namely $v := \gamma P v + g$, is equivalent to a Richardson relaxation of the corresponding linear system. It is well known that standard value iteration is slow to converge when $\gamma\mathbf{P}$ has an eigenvalue close to the unit disk, namely γ is close to 1, and this is exactly when that we expect AMG (and Multigrid in general) to provide a significant improvement.

Similarly, we can apply AMG to solve (2), with $\mathbf{A}$ and $\mathbf{b}$ as defined in (3). This is the starting point for the multigrid learning algorithms of the next section.

In the special case when the transition probability matrix P is symmetric, the matrix $A = I - \gamma P$ turns out to be an M-matrix with strictly dominant diagonal, a case to which AMG theory nicely applies. Note also that P is typically sparse in practical problems, hence so is A, a property which is important for Multigrid efficiency. However, since P is hardly ever symmetric, theoretical performance bounds are not readily available. Nonetheless, standard AMG algorithms can be applied to the non-symmetric case without modification, and practical experience shows that they perform well even when symmetry is violated (e.g., Stüben, 2001, p. 518). It should also be noted that convergence to the exact solution can always be enforced, simply by "turning off" the coarse grid corrections at some point, or by monitoring the error reduction as done in other hierarchical schemes (Schweitzer *et al.*, 1985; Bertsekas & Castañon, 1989). For the problems we tested, unforced convergence was always obtained.

5. Multigrid Temporal Difference Learning

To motivate the proposed Multigrid enhancement to TD(λ), we briefly consider the convergence of the mean of the parameter vector $\mathbb{E}\{\theta_t\}$, denoted $\bar{\theta}_t$. The stochastic dynamics of the TD(λ) algorithm, as derived in (Tsitsiklis & Van Roy, 1997), may be asymptotically approximated by $\bar{\theta}_{t+1} = \bar{\theta}_t + \alpha_t(b - A\bar{\theta}_t)$, where A and b are defined in (3). The error $\bar{\mathbf{e}}_t = \theta^* - \bar{\theta}_t$ relative to the fixed point $\theta^* = A^{-1}b$ satisfies $\bar{e}_{t+1} = (I - \alpha_t A)\bar{e}_t$. TD($\lambda$) may thus be interpreted as a stochastic smoother of the error. Multigrid is therefore a natural candidate for speeding up its convergence.

5.1. The SeqMGTD(λ) Algorithm

Our first algorithm mimics the V-cycle of the Multigrid algorithm as described in Section 2. We assume that we are given an initial (fine-level) set of K basis functions, with corresponding feature vectors $\phi_0(s) = \phi(s)$, as well as a set of interpolators $I_{\ell+1}^{\ell}$ and restrictors $I_{\ell}^{\ell+1}$. We then recursively define feature vectors for all levels by

$$\phi_{\ell+1}(s)^T = \phi_\ell I_{\ell+1}^{\ell}\,. \tag{4}$$

The algorithm is started by the function call $\theta_0 := \text{seqMGTD}(\theta_0, \ell = 0)$, where θ_0 is an initial guess.

The algorithm requires some switching criterion for the pre- and post-iterates. In the reported experiments we used the simplest rule of switching after a fixed number of iterations. Another reasonable option would be to increase the iteration count per level as the gain parameter decreases. The algorithm is started by the function call $\theta_0 := \text{seqMGTD}(\theta_0, \ell = 0)$, where θ_0 is an initial guess. The estimated value function at the end of a complete cycle is given by $v(s) = \phi_0(s)^T\theta_0$. The value function for intermediate times while level ℓ is completed is given more accurately by $v(s) = \sum_{m=0}^{\ell} \phi_m(s)^T\theta_m$.

SeqMGTD($\theta^0_\ell, \ell, \theta_0, \theta_1, \ldots, \theta_{\ell-1}$)

1. **Initialize level correction:** $\theta_\ell := \theta^0_\ell, z_t := 0$
2. **Pre-iterate at level ℓ with residual rewards:**
   - 2.1 Observe the transition $s_t \to s_{t+1}$ and the reward g_t at time t.
   - 2.2 Update the eligibility traces $z_\ell := \lambda\gamma z_\ell + \phi_\ell(s_t)$
   - 2.3 Sample the residual $r_\ell := g_t - \sum_{m=0}^{\ell-1} \Big(\phi_m(s_t) - \gamma\phi_m(s_{t+1})\Big)^T \theta_m$
   - 2.4 Calculate the temporal difference $d_\ell = r_\ell - \Big(\phi_\ell(s_t) - \gamma\phi_\ell(s_t+1)\Big)^T \phi_\ell$
   - 2.5 Update $\theta_\ell := \theta_\ell + \alpha_{\ell,t}\mathbf{z}_\ell d_\ell$
   - 2.6 If the switching criterion is met then continue, otherwise repeat from 2.1.
3. **Apply coarse grid correction:** If $\ell \neq \ell_{\max}$,
   - 3.1 Recursive call $\theta_{\ell+1} :=$ MGTD($\theta^0_{\ell+1} = 0, \ell+1, \theta_0, \theta_1, \ldots, \theta_\ell$)
   - 3.2 Correction using the interpolated error $\theta_\ell := \theta_\ell + I^\ell_{\ell+1}\theta_{\ell+1}$
4. **Post-iterate:** repeat step 2 until meeting the switching criterion.
5. **Return** θ_t.

Table 1: Sequential Multigrid TD(λ) at level ℓ

To understand this algorithm, note that the Multigrid algorithm (as presented in Section 2) at level ℓ is aimed at the solution of the equation $A_\ell x_\ell = r_\ell$, where r_ℓ is defined recursively via $r_\ell = I^\ell_{\ell-1}(r_{\ell-1} - A_{\ell-1}x_{\ell-1})$. In the RL context, this equation cannot be represented explicitly since the matrices A_ℓ and r_ℓ are unavailable. We resolve this problem by using an appropriate TD(λ) type iteration that serves as the iterative smoother for that level. First, the interpolated residual r_t is *sampled* as step 2.3, and serves as the driving reward signal for that stage. The TD(λ) algorithm then proceeds with the level-ℓ basis functions. A transition to level $\ell+1$ then takes place for the purpose of coarse grid correction, intended to accelerate the convergence of the smoothed error at level ℓ.

At the coarsest level ($\ell = \ell_{\max}$) step 3 and 4 should be skipped. Alternatively, the TD(λ) iteration at that stage may be replaced by another learning algorithm such as LSTD(λ) ((Boyan, 2002) or λ-LSPE (Nedić & Bertsekas, 2003).

The algorithm is sequential in nature, as the different levels operate on non-overlapping time intervals. This implies, in particular, that to make full use of data points at each level requires data reuse, or *experience replay*. Resetting of the eligibility traces at the beginning of each stage is not necessary if temporal continuity of the data samples is maintained in subsequent activations of the same level.

It may be verified (Ziv, 2004) that each level *in isolation* approaches the solution of the desired equation at that level. Convergence of the overall algorithm cannot be established in general, as even the basic AMG algorithm is not guaranteed to converge without further restrictions on the system matrices or inter-level operators. However, with a bounded number of smoothing iterates per level and diminishing gain, more complete results may be obtained. This is shown for the following variant of the algorithm.

5.2. The SimMGTD(λ) Algorithm

We consider a variant of the last algorithm which proceeds simultaneously at all levels, thereby eliminating the requirement for data reuse. Moreover, for this variant a convergence analysis of overall algorithm is provided. The algorithm is shown in the next table.

The SimMGTD(λ) algorithm has the following distinctive features: (1) All parameters except θ_0 are reset to 0 before each TD(λ) iteration. (2) The same temporal difference signal d_0 is used at all levels. Thus, the temporal difference updates at all levels are carried out simultaneously and instantaneously, and no coarse-level parameters need to be retained for the next iteration. As before, the value function estimate is $v(s) = \phi_0(s)^T\theta_0$. The convergence properties of this algorithm are summarized as follows (Ziv, 2004).

Theorem 2 *Assume that the SimMGTD(λ) algorithm is implemented with proportional gains, namely $\alpha_{\ell,t} = \beta_\ell\alpha_t$ for some non-negative constant β_ℓ, with $\beta_0 > 0$. Assume further that the conditions of Theorem 1 hold with respect to the level-0 basis functions $\phi(s) = \phi_0(s)$ and the above gain factors α_t. Then θ_ℓ*

converges (w.p. 1) to the same limit point as the standard TD(λ) algorithm at the finest level, namely to the solution of equation (3).

A. **Basic loop:**
   1. Initialize: Choose initial θ_0, set $z_t := 0$ for all ℓ
   2. Observe the transition $s_t \to s_{t+1}$ and the reward g_t at time t.
   3. Calculate the fine-level temporal difference: $d_0 := g_t - \Big(\phi_0(s_t) - \gamma\phi_0(s_{t+1})\Big)^T \theta_0$
   4. Update θ_0: $\theta_0 := \text{MGTD}(\theta_0, \ell = 0)$
   5. $t := t + 1$; goto A.2.

B. MGTD(θ_ℓ, ℓ) (recursive function)
   1. Update eligibility trace: $z_\ell := \lambda\gamma z_\ell + \phi_\ell(s_t)$
   2. TD(λ) iteration: $\theta_\ell := \theta_\ell + \alpha_{\ell,t} z_\ell d_0$
   3. **Coarse grid correction:**
      3.1 Recursive call: $\theta_{\ell+1} := \text{MGTD}(\theta_{\ell+1} = 0, \ell + 1)$
      3.2 Correction: $\theta_\ell := \theta_\ell + I^\ell_{\ell+1}\theta_{\ell+1}$
   4. Return θ_ℓ.

Table 2: Simultaneous Multigrid TD(λ)

6. Simulation

We next present a simulation experiment, which is meant to illustrate in an idealized problem setting the potential benefits of the proposed algorithms. The test-bed problem we consider is a 1-D random walk described in Figure 1. This Markov chain has N states, ordered on a 1-D line. Transition probabilities and rewards from inner states and edge states are defined in the figure, and a discount factor of $\gamma = \sqrt[N-1]{0.5}$ is chosen, so that the effective discount factor for a complete sweep of the state is space 0.5. This problem is similar to the hop-world problem in Xu *et al.* (2002), and should provide favorable conditions for performance improvement by multigrid methods, due to the local nature of the transition structure.

In the setup phase of AMG we used two interpolation methods, as discusses in Section 2: the Ruge-Stüben method and state aggregation. In Figure 2 we show results for the (non-learning) value iteration scheme of

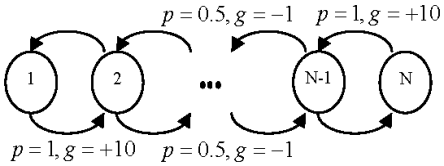


Figure 1. The 1-D random walk problem.

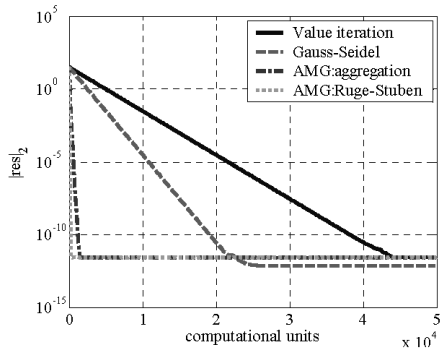


Figure 2. Convergence curves for the 1-D random walk problem with 1000 states. The AMG methods use 6 grid levels, with one pre- and post-smoothing iteration for seqMGTD.

Section 4. One computational unit equals the number of mathematical operations required for a single sweep of standard value iteration. The computational effort required to reach a residual error norm of 10^{-10} was 38203 for standard value iteration, 19103 for the Gauss Seidel variant, 1174 for AMG with state aggregation, and 23 for AMG with Ruge-Stüben interpolation.

We next consider the Multigrid learning algorithms for the same problem, this time with 256 states. The trivial basis functions were used in the first level (namely $\Phi_0 = 1$). For the purpose of the setup phase the full model was made available, Ruge-Stüben interpolation was employed. We used TD(0) at all levels (including the coarsest one) and a constant gain of $\alpha = 0.1$ throughout. In SeqMGTD, we switched levels every 5000 samples (which accounts for the periodic ripple of the corresponding graph). The norm of the error

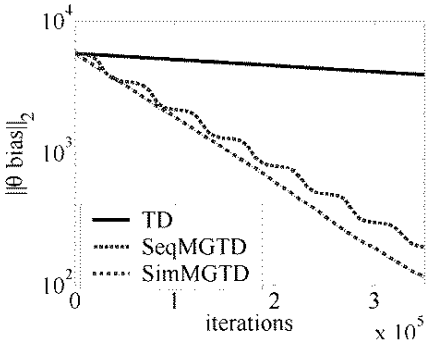


Figure 3. Learning curves for the random walk problem with 256 states. AMG methods use 6 grid levels. Each curve is an average of 5 Monte-Carlo runs.

in the parameter vector (relative to its target value) is plotted in Figure 3 as a function of the number of iterations. The number of iterations (in thousands) required for reducing the error norm by half is 654 for standard TD, 76 for SeqMGTD, and 62 for simMGTD.

In both cases, AMG shows at least order of magnitude improvement relative to standard iteration methods.

7. Concluding Remarks

Multigrid methods are a major tool in computational mathematics for speeding up the convergence of iterative methods. As such, its interaction with dynamic programming, and with RL in particular, seems natural. In this paper we have outlined some specific ways in which Multigrid might be combined with temporal difference learning, in order to speed up its convergence.

Several issues remain concerning the possible application of the proposed algorithms. A central question is how to set up an effective multigrid hierarchy, namely the coarse level equations and inter-level operators. In many cases the (geometric) structure of the state space directly suggests a reasonable selection of state aggregates. When this is not the case, AMG suggests effective methods for the automatic creation of the multigrid hierarchy at the setup phase; however, the extension of these methods to the learning scenario is yet to be explored. Other issues relate to optimization of various parts of the iterative algorithms, such as the choice of the relative gains at the different levels and the switching rules between levels. Additional experimental work is obviously required to evaluate the overall efficacy of these algorithms, along with further theoretical results.

From another viewpoint, we observe that AMG is a bottom up approach which builds coarse bases from finer ones. Applying this process to the full state space, for example, may lead to a scheme for constructing effective basis functions, based on the considerable theoretical and practical insight of AMG research.

Acknowledgments

We would like to thank Irad Yavne for his invaluable guidance and advice on Multigrid methods.

References

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems: Theory and Applications*, *13*, 41–77.

Bertsekas, D. P., & Castañon, D. (1989). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Trans. Automat. Contr.*, *34*, 589–598.

Boutillier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: structural assumptions and computational leverage. *Journal of Intelligence Research*, *11*, 1–94.

Boyan, J. A. (2002). Technical update: least squares temporal difference learning. *Machine Learning*, *49*, 233–246.

Brandt, A., McCormick, S. F., & Ruge, J. (1984). Algebraic multigrid (AMG) for sparse matrix equations. In D. J. Evans (Ed.), *Sparsity and it applications*, 257–284. Cambridge University Press.

Briggs, W. L., Henson, V. E., & McCormick, S. F. (2000). *A multigrid tutorial*. Philadelphia, MA: Siam. 2 edition.

Nedić, A., & Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications*, *13*, 79–110.

Schweitzer, P. J., Puterman, M. L., & Kindle, K. W. (1985). Iterative aggregation-disaggregation procedures for discounted semi-Markov reward processes. *Operations Research*, *33*, 589–605.

Stüben, K. (2001). An introduction to algebraic multigrid. Appendix A in Trottenberg et al.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Trottenberg, U., Oosterlee, C., & Schüller, A. (2001). *Multigrid*. San Diego: Academic Press.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

Xu, X., He, H., & Hu, D. (2002). Efficient reinforcement learning using recursive least-squares methods. *Journal of Artificial Intelligence Research*, *16*, 259–292.

Ziv, O. (2004). Algebraic multigrid for reinforcement learning. Tecnical report (thesis draft), August 2004, available at http://www.ee.technion.ac.il/shimkin/preprints.htm.