A Bayesian Approach to Finding Compact Representations for Reinforcement Learning

Alborz Geramifard[†], **Stefanie Tellex**[*], **David Wingate**[†], **Nicholas Roy**[*], **Jonathan How**[†]

[†]{AGF,WINGATED,JHOW}@MIT.EDU

[*]{STEFIE10,NICKROY}@CSAIL.MIT.EDU

Abstract

Feature-based function approximation methods have been applied to reinforcement learning to learn policies in a data-efficient way, even when the learner may not have visited all states during training. For these methods to work, it is important to identify the *right* set of features in order to reduce over-fitting, enable efficient learning, and provide insight into the structure of the problem. In this paper, we propose a Bayesian method for reinforcement learning that finds a policy by identifying a compact set of high-performing features. Empirical results in classic RL domains demonstrate that our algorithm learns concise representations which focus representational resources on regions of the state space that are necessary for good performance.

Keywords: reinforcement learning, representation learning, policy iteration, Metropolis-Hastings

1. Introduction

For many learning problems, the choice of representation is critical to an algorithm's performance: a good representation can make learning easy, but a poor one can make learning impossible. In reinforcement learning (RL), this problem is most pronounced when designing function approximators to solve complex problems: underpowered representations require a small number of samples to fit yet they often lead to poor polices. Expressive representations provide good approximations, yet they often require a plethora of training data and the learned representation may not provide insight into the structure of the problem. Hence, finding a concise set of features that leads to a good approximation is of high importance when samples and computation are limited.

While many function approximators have been proposed, linear value function approximators have enjoyed special attention because of a combination of analytic tractability and good empirical performance (Sutton, 1988; Bradtke and Barto, 1996; Lagoudakis and Parr, 2003; Bowling et al., 2008; Sutton et al., 2009). In any linear architecture, a function $f(x)$ is approximated by a linear combination of features: $f(x) \approx w^T \phi(x)$, where $\phi(x)$ is known as a *feature extractor* and w is a corresponding weight vector. In order for the system to avoid over-fitting and generalize to previously unseen states, a compact set of predictive features must be identified. Previous approaches have identified heuristics for expanding the feature representation (Geramifard et al., 2011; Sutton and Whitehead, 1993; Mahadevan, 2005; Fahlman and Lebiere, 1991), but these methods are not biased towards learning a compact set of features that captures only the essential elements necessary to solve the planning problem.

This paper proposes a Bayesian approach to feature construction which addresses these concerns. We use the performance of a particular representation as a likelihood term, and combine it with a prior that favors simple representations. Inference in the resulting probabilistic model jointly optimizes both terms, identifying a set of features which is simultaneously compact and performs well. For example, for the inverted pendulum problem, our algorithm identifies a single extra feature, which, when added to the representation, enables the system to perform optimally.

2. Background

Reinforcement learning (RL) is a powerful framework for sequential decision making in which an agent interacts with an environment on every time step. The environment is often modeled using a Markov Decision Process (MDP) which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}, \gamma)$, where $\mathcal{S}$ identifies the finite set of states, $\mathcal{A}$ corresponds to the finite set of actions, $\mathcal{P}^a_{ss'}$ dictates the transition probability from state s to state s' when taking action a, $\mathcal{R}^a_{ss'}$ is the corresponding reward along the way, and $\gamma \in [0, 1]$ is a discount factor emphasizing the relative significance of immediate rewards versus feature rewards.[1] A trajectory of experience is identified by the sequence $s_0, a_0, r_1, s_1, a_1, r_2, \cdots$, where at time i the agent in state s_i took action a_i, received reward r_{i+1}, and transited to state s_{i+1}. The behavior of the agent is captured through the notion of policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ governing the probability of taking each action in each state. We limit our attention to deterministic policies mapping each state to one action. The value of a state given policy π is defined as the expected cumulative discounted rewards obtained starting the sequence from s and following π thereafter:

$$V^\pi(s) \quad = \quad E_\pi\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s\right]$$

Similarly the value of a state-action pair is defined as:

$$Q^\pi(s, a) = E_\pi\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s, a_0 = a,\right]$$

The objective is to find the optimal policy defined as:

$$\pi^*(s) \quad = \quad \operatorname*{argmax}_a Q^{\pi^*}(s, a).$$

One popular thrust of online reinforcement learning methods such as SARSA (Rummery and Niranjan, 1994) and Q-Learning (Watkins and Dayan, 1992) tackle the problem by updating the estimated value of a state based on *temporal difference error* (TD-error), while acting mostly greedy with respect to the estimated values. TD-error is defined as

$$\delta_t(Q^\pi) = r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t).$$

One of the main challenges facing researchers is that most realistic domains consist of large state spaces and continuous state variables. Function approximators have been used as a machinery to overcome these obstacles, enabling an agent to generalize its experience in order to act appropriately in states it may have never previously encountered during training. Linear function approximators, which are the focus of this paper, have been favored due to their theoretical properties and low computational complexities (Sutton, 1996; Tsitsiklis and Van Roy, 1997; Geramifard et al., 2006). Using a linear function approximation $Q^\pi(s, a)$ is approximated by $w^T\phi(s, a)$ where $\phi : \mathcal{S} \times \mathcal{A} \to \Re^n$ is the mapping function and w is the weight vector. For simplicity we call $\phi(s, a)$ the feature vector and each element of the vector a feature.

Finding a suitable mapping function is one of the critical elements to obtain an adept policy. Early studies on random feature generation methods have shown promising directions on expanding the representation using some basic set of features (Sutton and Whitehead, 1993). Representational Policy Iteration (RPI) (Mahadevan, 2005) is another approach for discovering task independent representations fusing the theory of smooth functions on a Riemannian manifold with the Least-Squares method. Another popular trend of methods migrated the idea of Cascade-Correlation (Fahlman and Lebiere, 1991) to the reinforcement learning realm using temporal difference learning (Rivest and Precup, 2003), approximate dynamic programming (Girgin and Preux, 2007), and LSPI (Girgin and Preux, 2008). (Geramifard et al., 2011) described a method for incrementally adding a feature which maximally reduces TD-error. However, none of these techniques facilitate a regularization scheme by which the designer incorporates his knowledge over the set of hypotheses.

From the Bayesian cognitive science community, Goodman et al. (2008) used a grammar-based induction scheme to learn human concepts in a supervised learning setting. In their approach new concepts (features) were derived from a limited set of initial propositions using a generative grammar. This work motivated us to revisit the representational expansion within the RL community from the Bayesian approach.

3. Our Approach

We adopt a Bayesian approach to find well performing policies. The core idea is to find a representation

1. $\gamma = 1$ is only valid for episodic tasks.

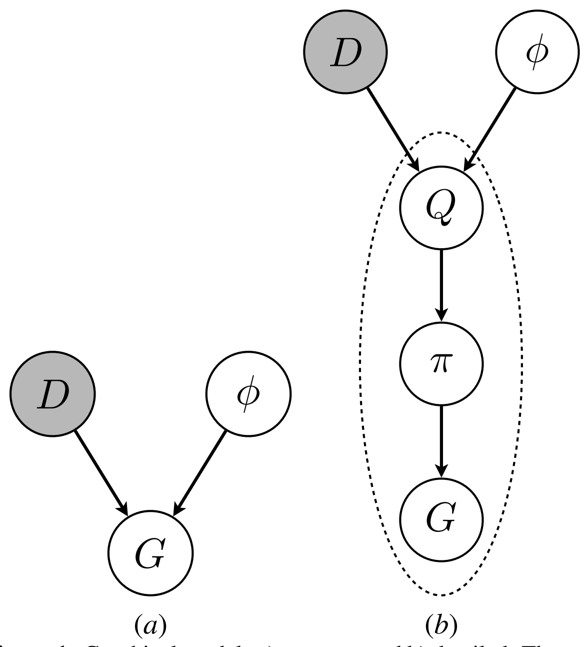


Figure 1: Graphical models a) compact and b) detailed. The gray node (Data) is given.

for which, given a dataset, the resulting policy is most likely to be optimal. For this purpose, we define variable $G \in \{0,1\}$ that indicates if a given policy is optimal.[2] Define D as our observed variable that is a set of interactions in the form of $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$. Hence our search for a good representation is formulated as:

$$\phi^* = \operatorname*{argmax}_{\phi} P(\phi|G,D). \tag{1}$$

Figure 1-(a) depicts the corresponding graphical model. D is filled to show that it is a known variable. We factor the distribution in order to break it down into a prior and a likelihood. The solution to our optimization problem in Equation 1 can be stated as finding the Maximum-a-Posteriori (MAP) solution to the following distribution:

$$\begin{aligned} P(\phi|G,D) &\propto P(G|\phi,D)P(\phi|D) \\ &\propto P(G|\phi,D)P(\phi). \end{aligned} \tag{2}$$

We define a family of representations by forming new features which are combinations of existing features using logical operators. This space is infinite, and could lead to very complex representations; we address this issue by defining a prior on representations that favors simplicity, so that concise representations should have a higher *a priori* probability compared to complex ones. The likelihood function is based on the performance of the representation at obtaining reward. Defining the likelihood requires first computing the value of each state, Q, using LSPI. Then the policy π is calculated as a function of Q. Finally G probes the quality of the computed policy. This chain of relations is captured in Figure 1-(b).

Since inference for Equation 2 is searching an exponential space, our algorithm samples representations from the posterior distribution using the Metropolis-Hastings algorithm. Because our algorithm integrates Metropolis-Hastings with LSPI, we call the algorithm Metropolis-Hastings Policy Iteration (MHPI).

3.1. Space of Representations

In this work, we assume the presence of a set of *primitive* binary features. Starting with the initial set of features, the system adds *extended* features, which are logical combinations of primitive features built using the $\wedge$ and $\vee$ operators. Each representation is maintained as a directed acyclic graph (DAG) structure, where nodes are features, and edges are logical operators. We use sparse binary features (*i.e.,* feature vectors with very limited non-zero values) to reduce the computational complexity of learning methods, a common practice within the RL community (Bowling and Veloso, 2002; Sherstov and Stone, 2005). While adding negation ($\neg$) to the set of operators expands the hypothesis space, it eliminates the sparsity characteristic, so we do not include it in the space of representations. Figure 2 (left) shows an example feature set, where each feature is marked with its corresponding index. The number of features for this set is 8, where 6 of the features are primitive and 2 extended features are: $f_8 = f_4 \wedge f_6$ and $f_7 = f_2 \vee f_3$. Notice that more complex extended features can be built on top of existing extended features (*e.g.,* $f_9 = f_1 \wedge f_7$). In order to discourage representations with complex structures, we adopted the following Poisson distribution akin to the work of Goodman et al. (2008) to mathematically encourage conciseness:

$$P(\phi) \propto \prod_{i=1}^{n} \frac{\lambda^{d_i} e^{-\lambda}}{d_i!},$$

where n is the total number of features and d_i is the depth of feature f_i in the DAG structure. Lower values of $\lambda > 0$ make complex DAG structures less and less likely.

2. From this point, we use G instead of $G = 1$ for brevity.

3.2. Likelihood

The likelihood function states the probably of a resulting policy with a fixed representation and set of data to be optimal. This term often encourages expressive representations. One approach is to relate the likelihood to the accuracy of the value function after policy iteration.[3] The main drawback of this approach is that the likelihood function encourages representations with good value function approximation, rather than good resulting policies. In general a greedy policy with respect to the the exact value function is guaranteed to be optimal (Sutton and Barto, 1998), but when the value function is approximated, more accurate value functions do not necessarily provide better policies as the greedy policy is related to the ranking of the values not their accuracies. Consequently, we adopt a likelihood function that has high values for high-performing policies, based on reward earned from the initial state:

$$P(G|\phi, D) \quad \propto \quad e^{\eta V^{\pi_i}(s_0)} \tag{3}$$

$\eta > 0$ is the distribution parameter. Higher values of η make well performing policies more likely to be optimal.

To obtain π_i and $V^{\pi_i}(s_0)$, we use the least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003) algorithm, with one modification. It is known that each iteration of LSPI does not necessarily improve the resulting policy. Hence on each iteration of LSPI, we test the performance of the policy using Monte-Carlo simulations. After all iterations, the highest performing policy and its performance is returned. Notice that if a simulation box is not present, off-policy evaluation techniques such as importance sampling (Sutton and Barto, 1998) and model-free Monte Carlo (Fonteneau et al., 2010) can be used.

3.3. Inference

So far, we have introduced a probability distribution over representations, where representations with the best trade-off between conciseness and expressivity are most likely. While $P(\phi|G, D)$ can be sampled for each representation, the shape of it is not known, posing a challenging for direct inference. To mitigate this problem, we use the Metropolis-Hastings (MH) (Hastings, 1970) algorithm.

3. Assuming each TD-error, δ_i is sampled independently from $\mathcal{N}(0, \sigma^2)$ then $P(D|\phi) = \Pi_{i=1}^{|D|} G(\delta_i; 0, \sigma^2)$, where G is the Gaussian probability density function.

```
Input: $\phi$, propose, $T$, $P(\phi|G, D)$, SampleSize
Output: Samples
foreach $i \in \{1, \cdots, \texttt{SampleSize}\}$ do
    Samples($i$) = $\phi$
    $\phi' \leftarrow$ propose($\phi$)
    if $rand < \min\left\{1, \frac{P(\phi'|G,D)T(\phi|\phi')}{P(\phi|G,D)T(\phi'|\phi)}\right\}$ then
        $\phi \leftarrow \phi'$
return Samples
```

Algorithm 1: Metropolis-Hastings Policy Iteration

MH is a Markov chain Monte Carlo method for sampling random variables for which the underlying distribution is not known, shown in Algorithm 1. The algorithm is seeded with the initial representation ϕ. The `propose` function generates a new candidate sample (ϕ') on each iteration. The T function is the transition probability; $T(\phi'|\phi)$ states the probability of the `propose` function generating ϕ' from ϕ.[4] The posterior distribution function, $P(\phi|G, D)$, is calculated using Equation 2. `SampleSize` states the number of samples generated through MH. On each iteration, the new candidate sample ϕ' is accepted scholastically based on a probability value that highlights the desirability of the sample. As the number of samples generated through MH goes to infinity, the distribution of the samples converges to P.

3.3.1. The `propose` Function

As stated in Section 3.1, we focus on set of features built using Boolean logic. Figure 2 explains three actions used in our `propose` function: add, $mutate$, and $remove$. The add action attaches a node with a random operator $\in \{\wedge, \vee\}$ connecting two children selected uniformly randomly from existing features (*e.g.,* f_9). The second action is to $mutate$ the operator in one of the extended nodes (*e.g.,* f_7). Because the representation is preserved as a DAG, the effect of mutation is propagated to related features on the top of the affected node, allowing fast exploration of the representation space. The last action is to $remove$ an extended feature. Note that in order to have the representation sound at all time, we only remove features from the top of the DAG structure with no parents (*i.e.,* "header" features). On each iteration of MH, the `propose` method selects one of the three actions with equal probability and generates a candidate representation.

4. In the original MH algorithm the convention is to use Q for the transition probability, yet this notation collides with the Q function of the RL framework. Hence we use T.

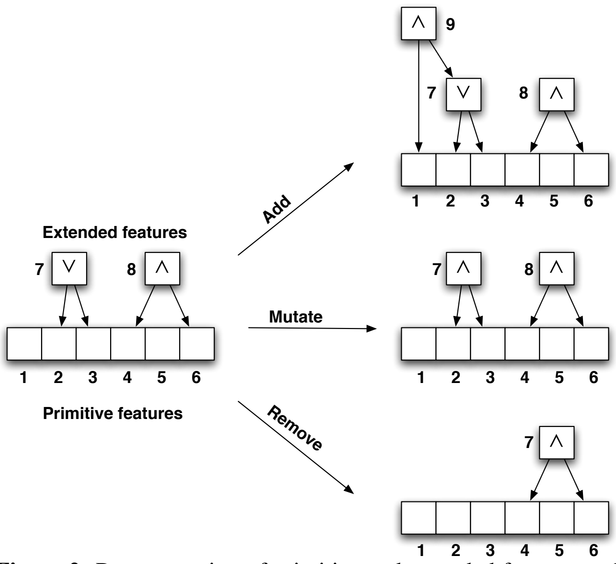


Figure 2: Representation of primitive and extended features and the possible outcomes of the `propose` function

3.3.2. The Transition Probability Function (T)

When the transition probability function is symmetric, it can be removed from the MH algorithm, as it is canceled out during the calculation. In our setting, however, T is not symmetric. Given that hypothesis ϕ has p primitive features, e extended features and h header features, and representation ϕ' proposed by taking action a from ϕ, the transition probability function is defined as:

$$T(\phi'|\phi) = \begin{cases} {}^{2(p+e)}/_{p+e-1} & a = add \\ {}^{1}/_{h} & a = remove \\ {}^{1}/_{e} & a = mutate \end{cases}$$

4. Empirical Results

In this section, we investigate the performance of running MHPI in the three domains: maze, BlocksWorld, and the inverted pendulum problem. For each domain samples were gathered by the SARSA (Rummery and Niranjan, 1994) algorithm using the initial feature representation with the learning rates generated from the following series:

$$\alpha_t = \alpha_0 \frac{N_0 + 1}{N_0 + \text{Episode\#}^{1.1}},$$

where N_0 was set to 100, and α_0 was initialized at 1 due to the short amount of interaction. For exploration, we chose the ϵ-greedy approach with $\epsilon = .1$ (*i.e.,* 10% chance of taking a random action on each time step). The λ parameter of the Poisson distribution was set to 0.01 while η for the exponential distribution was set to 1. The initial representation used for MH included all basic features. Additionally $\phi(s, a)$ was built by copying $\phi(s)$ vector into the corresponding action slot. Therefore $\phi(s, a)$ has $|A|$ times more features compared to $\phi(s)$. For LSPI, we limited the number of policy iterations to 5, while the value of the initial state for each policy, $V^{\pi_i}(s_0)$, was evaluated by a single Monte-Carlo run.

Maze Figure 3-(a) shows a simple 11×11 navigation problem where the initial state is on the top left corner of the maze ($\circ$) and the goal is at the bottom right corner of the maze ($\star$). Light blue cells indicate blocked areas. The action set consist of one step moves along the four cardinal directions. Actions are noiseless and possible if the destination cell is not blocked. Reward is $-.001$ for all interactions except the move leading to the goal with reward of $+1$. The episodic task is terminated if goal is reached, or 100 steps is passed. There were 22 initial features used for $\phi(s)$ corresponding to 11 rows and 11 columns of the maze. γ was set to 1.

We used 200 samples through 2 episodes, gathered in the domain using SARSA. The agent reached the goal in the first episode following the top right corner of the middle blocked square. The second episode failed as the agent struggled behind the blocked area on the bottom. Figure 3-(b) shows the distribution of the representation sizes sampled along 1,000 iterations of the MH algorithm, while Figure 3-(c) shows the corresponding performance of the sampled representations. The distribution together with the performance measure suggest that a desirable representation should have 3 extended features. After 100 iterations all sampled hypotheses were expressive enough to solve the task. It is interesting to see how Occam's Razor is being carried away through the whole process. The MH algorithm spent most of its time exploring various hypotheses with 3 extended features while the more complicated representations were of less interest as they provided the same performance (*i.e.,* likelihood) yet had lower prior.

Figure 3-(d) shows the value function (green indicates positive, white represents zero, and red stands for blocked areas) and the corresponding policy (arrows) for the best performing representation. This representation had 3 extended features: $(X = 2 \wedge Y = 11)$, $(X = 3 \wedge Y = 6)$, and $(X = 2 \wedge Y = 8)$, where X is the row number and Y is the column number. Notice that the policy guides the agent successfully from the starting point to the goal on the shortest path.

At the same the policy is not optimal in the whole state space. This observation which is aligned with our initial statement is due to two factors. First samples gathered were sparse and did not cover the whole state space. Hence the algorithm generalizes the values incorrectly in some parts of the state space it has not been exposed to (*e.g.,* ignoring the presence of a blockade on the last column). Secondly, the performance measure was solely based on the execution of the policies from the initial state. If a globally optimal policy is desired, V^{π_i} in Equation 3 should be evaluated from various initial states. In the next experiment, we probed a larger domain that also includes uncertainty.

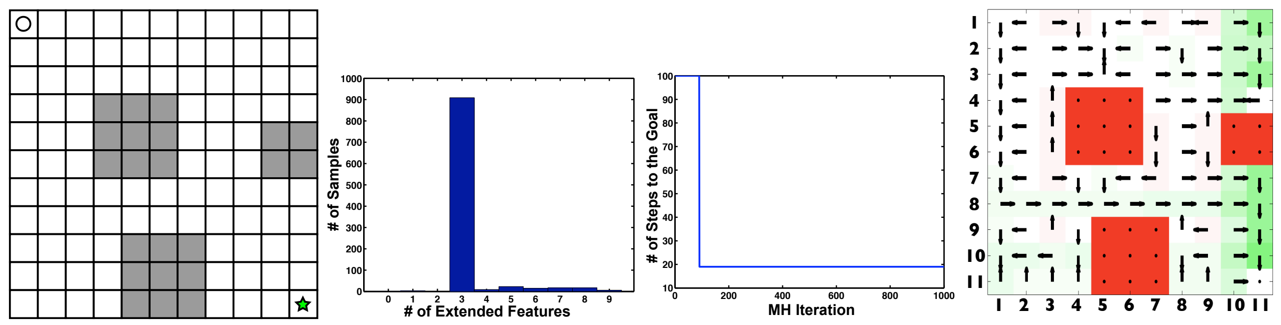
BlocksWorld Figure 4-(a) depicts the classical BlocksWorld domain. The task starts with 4 blocks laying on the table. The episodic task is to build a tower out of all blocks with a predefined color order. Each episode is finished after 100 steps. In each state, the agent can take any clear block (*i.e.,* a block with no other block on top of it) and attempt to move it on the top of any other clear blocks or table. Any move involves 20% chance of failure resulting in dropping the block on the table. $\phi(s)$ was derived directly from the logical representation of $on(A, B)$ resulting in 16 basic features. The reward function was identical to the maze domain.

In this domain, we increased the number of samples to $1,000$ as the environment was noisy and seeing the goal state was more challenging than the previous deterministic task. The agent finished making the tower only once. Figure 4-(b) shows the distribution of the representation size after running MH, while Figure 4-(c) shows the corresponding performance through the number of steps to reach the goal. It is interesting to see that the performance started from the top which means the primitive representation was not expressive enough to solve the task, yet after few iterations the extended features made the task learnable. The small perturbation on the performance graph is due to the stochasticity, causing some trajectories to be longer than the others. According to MH, representations with 4 and 5 extended features were most likely even though fewer extended feature could solve the task. We conjecture that if our agent gets lucky and does not drop blocks by accident, it does not need more extended features to solve the puzzle. On the other hand, if blocks are dropped during the movement which happens frequently, the agent experiences new parts of the state space. Hence it needs more features to differentiate the value function correctly. In order to verify our hypothesis, we plotted the average number of steps it took the agent to build the tower based on the size of the representation. On failed trials the episode cap (*i.e.,* 100 steps) were accumulated. Figure 4-(d) shows the corresponding result including standard error bars with 95% confidence.[5] Adding one extended feature rendered the task learnable, yet the resulting policies were not robust to the stochasticity as the error bar highlights. Overall, extended features reduced variance and improved the result as long as they were less than 6. The optimal expected number of steps for this problem is 3.75. Adding more than 5 extended features increased variance and increased the expected number of steps due to overfitting. This trend coincides with the sample distribution shown in Figure 4-(d). Next, we probe MHPI in a domain with continuous state variables.

Inverted Pendulum Figure 5-(a) depicts the inverted pendulum domain based on the previous work of Lagoudakis and Parr (2003). The episodic task is to balance the pendulum up right as long as possible. Each episode is finished when the pendulum hits the ground or reach the cap of $3,000$ steps. The state of the system is defined by the angle and angular velocity of the pendulum, $[\theta, \dot{\theta}]$. Both dimensions of the initial state were sampled from the uniform distribution between $[-0.2, +0.2]$ on each episode. The set of actions were limited to three values of force: $\{-50, 0, +50\} + \omega$ where ω is a uniform noise between $[-10, 10]$. The reward was $+0.01$ for all steps except for the time that the pendulum hits the ground resulting in -1 reward. Notice that we deviated from the reward function in the original work in order to differentiate between the performance of different representations. Otherwise LSPI would have returned -1 for all representations incapable of balancing the pendulum for $3,000$ steps. γ was set to 0.95. Basic features were generated by discretizing each dimension into 21 buckets separately translating into 42 initial features.

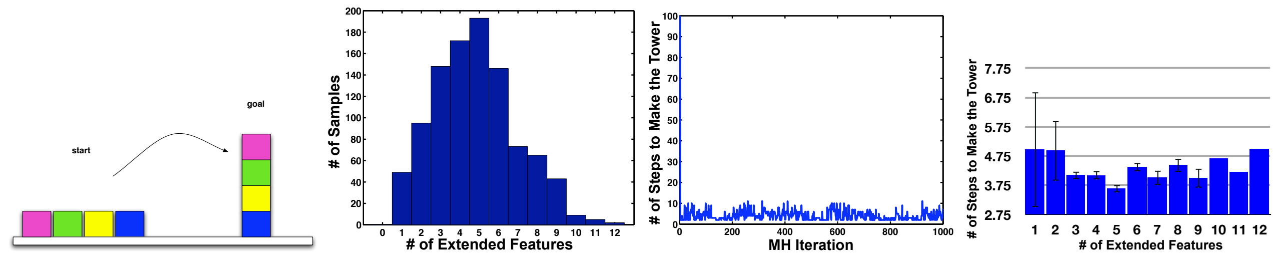
We gathered $1,000$ steps of experience through 70 trajectories. Figure 5-(b) shows the histogram of the extended feature size through 500 iterations of MHPI, while Figure 5-(c) depicts the corresponding perfor-

5. For representations with more than 9 extended features, we did not have more than 30 samples, hence we excluded the standard error.



(a) Domain (b) Posterior Distribution (c) Sampled Performance (d) Resulting Policy

Figure 3: Maze domain empirical results



(a) Domain (b) Posterior Distribution (c) Sampled Performance (d) Performance Dist.

Figure 4: BlocksWorld

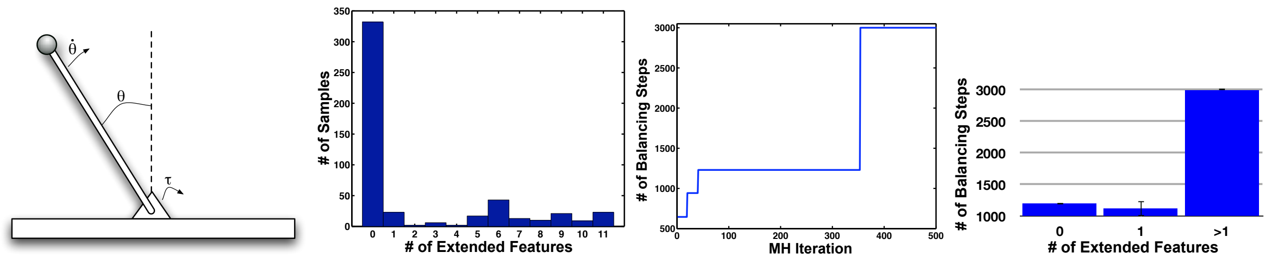
mance along each iteration. Unlike other domains that more features often helped the performance early on. In this domain irrelevant features dropped the performance resulting in MH to reject them. This process took a while until interesting features started to emerge. This effect is usually avoided by setting a burn-in value discarding limited number of initial samples in the MH setting. Yet we added this data to highlight the fact that expanding the representation arbitrary does not necessarily improve the performance in light of limited data. Figure 5-(d) shows the performance of the representations based on the number of extended features. In our experiments, while adding most extended features hurt the performance, the extended feature $(-\frac{\pi}{21} \leq \theta < 0) \wedge (0.4 \leq \dot{\theta} < 0.6)$ enabled the agent to complete the task successfully. This feature identifies an intuitive situation where the pendulum is almost balanced with a velocity on the opposite direction, which would be often visited. This is very interesting results, because out of all possible correlations among the initial features (21×21), capturing one such intuitive feature made the task solvable with very limited amount of data.

In our work, we found that the adjustment of priors played a critical role on the success of MHPI as priors compete against the performance of the resulting policies. Also we found MHPI to be robust in handling stochastic domains. For exampling adding 20% noise to the movement of the agent in the maze domain did not change the performance noticeably.

5. Conclusion

This paper introduces a Bayesian approach for finding concise yet expressive representations for solving MDPs. We introduced MHPI, a new RL technique that builds new representations from limited number of simple features that perform well. Our approach uses a prior distribution that encourages representation simplicity, and a likelihood function based on LSPI to encourage representations that lead to capable policies. MHPI samples representations from the resulting posterior distribution. Although, the idea of MHPI is general, in our implementation, we narrowed the representation space to DAG structures on primitive binary features. The empirical results show that MHPI finds simple yet effective representations for three classical RL problems.

There are immediate visible expansions to this work. In our implementation, we excluded the samples generated during the performance test in order to take advantage of caching old representation evaluations. One can use such samples along the way while being aware of the increase to the runtime complexity. Another extension is to relax the need of the simulation box in LSPI by measuring the performance using off-



(*a*) Domain (*b*) Posterior Distribution (*c*) Performance (*d*) Performance Dist.

Figure 5: Inverted pendulum

policy evaluation techniques such as importance sampling (Sutton and Barto, 1998) and model-free Monte Carlo (Fonteneau et al., 2010).

References

M. Bowling and M. Veloso. Scalable learning in stochastic games, 2002.

Michael Bowling, Alborz Geramifard, and David Wingate. Sigma point policy iteration. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 379–386, 2008.

S. Bradtke and A. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

Scott E. Fahlman and Christian Lebiere. *The Cascade-Correlation Learning Architecture*, 1991.

Raphael Fonteneau, Susan A. Murphy, Louis Wehenkel, and Damien Ernst. Model-free monte carlo-like policy evaluation. *Journal of Machine Learning Research - Proceedings Track*, 9:217–224, 2010.

Alborz Geramifard, Michael Bowling, and Richard S. Sutton. Incremental least-square temporal difference learning. In *The Twenty-first National Conference on Artificial Intelligence (AAAI)*, pages 356–361, 2006.

Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan How. Online discovery of feature dependencies. In Lise Getoor and Tobias Scheffer, editors, *International Conference on Machine Learning (ICML)*, pages 881–888, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

Sertan Girgin and Philippe Preux. Feature Discovery in Reinforcement Learning using Genetic Programming. Research Report RR-6358, INRIA, 2007.

Sertan Girgin and Philippe Preux. Basis function construction in reinforcement learning using cascade-correlation learning architecture. In *ICMLA '08: Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications*, pages 75–82, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3495-4. doi: http://dx.doi.org/10.1109/ICMLA.2008.24.

N. D. Goodman, J. B. Tenenbaum, T. L. Griffiths, and J. Feldman. Compositionality in rational analysis: Grammar-based induction for concept learning. In M. Oaksford and N. Chater, editors, *The probabilistic mind: Prospects for Bayesian cognitive science*, 2008.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970. doi: 10.1093/biomet/57.1.97.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

Sridhar Mahadevan. Representation policy iteration. *Proceedings of the 21st International Conference on Uncertainty in Artificial Intelligence*, 2005.

Franois Rivest and Doina Precup. Combining td-learning with cascade-correlation networks. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 632–639. AAAI Press, 2003.

G. A. Rummery and M. Niranjan. Online q-learning using connectionist systems (tech. rep. no. cued/f-infeng/tr 166). *Cambridge University Engineering Department*, 1994.

Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In J.-D. Zucker and I. Saitta, editors, *SARA 2005*, volume 3607 of *Lecture Notes in Artificial Intelligence*, pages 194–205. Springer Verlag, Berlin, 2005.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. The MIT Press, 1996.

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

Richard S. Sutton and Steven D. Whitehead. Online learning with random representations. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 314–321. Morgan Kaufmann, 1993.

Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: http://doi.acm.org/10.1145/1553374.1553501.

John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. doi: 10.1007/BF00992698.