Goal-Directed Online Learning of Predictive Models

Sylvie C.W. Ong, Yuri Grinberg, and Joelle Pineau

School of Computer Science
McGill University, Montreal, Canada

Abstract. We present an algorithmic approach for integrated learning and planning in predictive representations. The approach extends earlier work on predictive state representations to the case of online exploration, by allowing exploration of the domain to proceed in a goal-directed fashion and thus be more efficient. Our algorithm interleaves online learning of the models, with estimation of the value function. The framework is applicable to a variety of important learning problems, including scenarios such as apprenticeship learning, model customization, and decision-making in non-stationary domains.

Keywords: predictive state representation, online learning, model-based reinforcement learning

1 Introduction

Reinforcement learning (RL) is the problem of learning how to behave so as to maximize long-term reward through interactions with a dynamic environment. In recent years, there has been an increased interest in the case of *batch* reinforcement learning, whereby the data used for learning how to behave are collected *a priori*. However in many domains, the case of *online* reinforcement learning is more relevant than the batch case. An additional complication arising in the online case is that the agent must explore its environment efficiently. Hence, one of the central issues in online reinforcement learning is the trade-off between exploration and exploitation. Generally speaking, to ensure efficiency in learning, there is a need for approaches that can achieve *goal-directed learning* from interaction [23].

Much of the prior work on reinforcement learning in unknown domains has focused on domains with full state observability [23]. The problem typically becomes much harder in partially observable environments, where the state of a dynamical system is in general a function of the entire history of actions and observations. Methods have been proposed for online learning in the partially observable Markov decision process (POMDP) [17, 20]. However these methods require a clear definition of the latent state variables of the system; this is not always easy to formalize in some decision-theoretic systems. For example, in human-robot interaction tasks, we typically need to handle partial state observability, yet it is often unclear what set of latent variables are appropriate for

capturing the human's intention, cognitive state, or attentional level, to name just a few relevant factors.

The predictive state representation (PSR) is a promising approach that forgoes the notion of underlying latent variables, and instead represents system state by the occurrence probabilities of future observation sequences, conditioned on future action sequences [13]. Since the representation is based entirely on observable quantities, in principle, predictive models should be easier to learn from data. PSRs have also been shown to have greater representational power than POMDPs [21]. Prior work on learning predictive models have mostly taken the approach of learning a complete model of the system with the objective of obtaining good predictive accuracy for all possible future behaviors, then planning with the learned model. Thus they either assume the training data is acquired through interaction with the environment using a purely exploratory policy [4], or else side-step the exploration problem by learning from a batch of pre-sampled action-observation trajectories [2].

Such approaches have several practical limitations. First, they assume an arbitrary division between the learning phase and acting phase, which implies that the cost of learning is unimportant. Second, when dealing with large domains (i.e. many observations and actions) and limited data, they tend to suffer from sparse parameter estimation and numerical instability (with some exception, e.g. [2]). Third, they don't naturally extend to handle non-stationary environments, unless a separate mechanism is involved to detect environment changes and initialize a new learning phase.

In this paper, we propose an online algorithm for learning predictive models of dynamical systems through *goal-directed* interaction. To our knowledge, this is the first approach for learning predictive models that integrates learning and planning in a way that explicitly tackles the problem of exploration and exploitation. Our approach is (loosely) based on the actor-critic framework. Model parameters are estimated in an online fashion, interleaved with steps of data gathering and policy optimization. The agent's behavior policy for interacting with the environment is derived from the most recently optimized policy, which in turn is obtained from planning with the current model and the data gathered thus far.

The contributions of this paper are primarily algorithmic and empirical (rather than theoretical). In contrast to previous approaches to predictive model learning, the objective is not to learn the complete model, but to only learn relevant aspects of the model for obtaining a good policy. Compared to prior work, our approach should therefore be more data efficient and thus more easily scalable to large action/observation domains, while having less empirical regret while learning. Since learning and planning are integrated, our approach should also more easily handle non-stationary environments. In preliminary investigations, we demonstrate the performance of our method on a robot navigation domain.

2 Predictive State Representations

A controlled, discrete-time, finite dynamical system generates observations from a set $\mathcal{O}$ in response to actions from a set $\mathcal{A}$. At each time step t, an agent interacting with the system takes an action $a_t \in \mathcal{A}$ and receives an observation $o_t \in \mathcal{O}$. A history, $h = a_1 o_1 a_2 o_2 \cdots a_t o_t$, is a sequence of past actions and observations, while a test, $\tau = a^1 o^1 a^2 o^2 \cdots a^k o^k$, is a sequence of actions and observations that may occur in the future. The prediction for test τ given history h, $p(\tau|h)$, is the conditional probability that the observation sequence in τ occurs, if the actions specified in τ are executed. The null test ε is the zero-length test. By definition, the prediction for ε is always 1[13].

Let T be the set of all tests and H the set of all histories. Given an ordering over H and T, we define the system-dynamics matrix, $\mathcal{D}$, with rows corresponding to histories and columns corresponding to tests. Matrix entries are the predictions of tests, i.e., $D_{i,j} = p(\tau_j|h_i)$ [21]. Suppose the matrix has a finite number of linearly independent columns, and we denote the set of tests corresponding to those columns as the core tests, Q. Then, the prediction of any test is the linear combination of the predictions of Q. Let $p(Q|h)$ be a vector of predictions for Q given history h, and m_τ be a vector of weights for test τ, then, for all possible tests τ, there exists a set of weights m_τ, such that $p(\tau|h) = m_\tau^\mathsf{T} p(Q|h)$. Thus, the vector $p(Q|h)$ is a sufficient statistic for history, i.e. it represents the system state, b. The set of all possible states form the PSR state-space, $\mathcal{B}$, and we refer to the state $b = p(Q|h)$, as the projection of history h on the PSR state-space. Define M_{ao} as the matrix with rows m_{aoq}^T, for all $q \in Q$. Given a history h, a new action a, and subsequent observation o, the PSR state vector is updated by,

$$p(Q|hao) = \frac{p(aoQ|h)}{p(ao|h)} = \frac{M_{ao} p(Q|h)}{m_{ao}^\mathsf{T} p(Q|h)} \; . \tag{1}$$

The projection of any history can be calculated by repeatedly applying the above equation, starting from the prediction vector for the empty history, $p(Q|\phi)$, which represents the initial system state, m_0.

A PSR model is completely specified by the core tests Q, weight vectors m_{aol}, for all $a \in A$, $o \in O$, $l \in Q \cup \{\varepsilon\}$, and m_0,. We denote these model parameters collectively as $\mathbf{M}$. The usual approach to model learning is to approximate the system-dynamics matrix, $\mathcal{D}$, through interactions with the system, i.e. by sampling action-observation trajectories, then use $\mathcal{D}$ to estimate the model $\mathbf{M}$[21].

3 Planning in PSRs

An agent taking action $a \in \mathcal{A}$ in a dynamic environment receives not just an observation $o \in \mathcal{O}$ but also a reward $r \in \mathcal{R}$. (Rewards are real numbers, however we assume that there is a finite set $\mathcal{R}$ of possible values for immediate rewards.) Denoting $R(b, a)$ as the expected immediate reward for action a at system state b, the goal of planning in PSRs for an infinite horizon problem is to find an optimal policy that maximizes the expected cumulative rewards $\mathrm{E}[\sum_t \gamma^t R(b_t, a_t)]$. Here,

γ is a discount factor that constrains the expected sum to be finite, $b_t = p(Q|h_t)$ is the system state and a_t the action, at time t.

In general, there are two approaches to planning in PSRs. One approach is to extend POMDP planning algorithms that exploit the fact that V^*, the value function for an optimal policy, can be approximated arbitrarily closely by a piecewise-linear, convex function of the probability distribution (or belief) over latent state variables. This approach represents a value function as a set of linear functions and performs value iteration on the set, generating a new set of linear functions at each iteration. The prediction vector $p(Q|h)$ is the state representation in PSRs that is equivalent to the belief state in POMDPs. It has been shown that for each action $a \in \mathcal{A}$, the expected immediate reward $R(b, a)$ is a linear function of the prediction vector $b = p(Q|h)$ and thus, just as in POMDPs, the optimal value function V^* is also a piecewise linear function of the PSR system state[12]. Previous work has used this idea for planning in PSRs, for both exact[12], as well as approximate planning[10, 11, 3].

An alternative approach is to extend function approximation algorithms for RL that were originally developed for fully observable systems. The system state is compactly represented by a vector of features instead of the state variable, and a mapping is learned from features to the value function, Q-function, or actions. The features are usually prespecified (for example, using domain knowledge) and the mapping function is usually parametric, so planning reduces to learning the parameter values of the mapping function. The prediction vector $p(Q|h)$ is the state representation in PSRs that is equivalent to the feature vector in fully observable systems. Previous work has extended RL methods such as Q-learning[12], SARSA[18] and Natural Actor-Critic[1], by learning mapping functions as described above, with the PSR system state taking the place of feature vectors. We adopt this latter approach due to its flexibility and scalability.

4 Online Reinforcement Learning with Predictive Models

We now present our algorithmic approach to model-based online RL in Algorithm 1. Just as in the PSR planning methods mentioned in Section 3, the algorithm does planning in the PSR state-space, and aims to learn a good policy as a function of the PSR system state.

4.1 Algorithm overview

Given a PSR model, $\mathbf{M}$, any history can be projected to the corresponding PSR state-space. To learn such a model, the usual steps mentioned in Section 2 are performed – sampling action-observation trajectories from the environment, approximating the system-dynamics matrix $\mathcal{D}$, and estimating the model $\mathbf{M}$. Unlike previous approaches however, our objective is online RL, not offline planning, thus model learning in our algorithm is goal-directed – trajectories are sampled so as to balance exploration of the environment with exploitation to obtain good rewards. To accomplish this, the algorithm interleaves optimizing

Algorithm 1 Goal-directed Online Model Learning

1: **input:** Initial behavior policy, π^e_{init}.
2: **initialization:** $i \leftarrow 1$; $\pi^e_i \leftarrow \pi^e_{init}$.
3: **repeat**
4: Sample $\bar{\mathcal{G}}_i$, a set of N_i trajectories from the environment (Algorithm 2, which requires as input, behaviour policy π^e_i and model $\mathbf{M}_{i-1}$.) Let $\mathcal{G}_i$ denote the set of trajectories sampled up to iteration i, i.e. $\mathcal{G}_i = \mathcal{G}_{i-1} \bigcup \bar{\mathcal{G}}_i$.
5: From $\mathcal{G}_i$, approximate the system-dynamics matrix $\mathcal{D}_i$, and estimate $\mathbf{M}_i$ (described in Section 4.2).
6: Optimize policy π_i with $\mathbf{M}_i$ and $\mathcal{G}_i$ (Algorithm 3).
7: $i \leftarrow i + 1$.
8: Update the behaviour policy π^e_i using π_{i-1}.
9: **until** stopping conditions are reached.
10: **output:** Predictive model $\mathbf{M}$, and its policy π.

a policy π using the current model $\mathbf{M}$ (line 6), and, sampling trajectories (line 4) using a behavior policy π^e derived from π (line 8). The model $\mathbf{M}$ is then updated based on the samples (line 5), and the process repeats. A key feature of the algorithm is that learning the model and planning with the model are integrated. If the process is interrupted or stopping conditions reached (line 9), the algorithm outputs both a model $\mathbf{M}$ and its policy π (line 10). This is in contrast to previous approaches which learn the model and then plan with it, as two separate and distinct processes.

The algorithm is a general framework with different possible choices for behaviour policies, and model learning and planning methods. For concreteness, we describe a particular implementation of these three steps.

The behaviour policy π^e is a stochastic policy which maps the system state b to a probability $\pi^e(b, a)$ for each action $a \in \mathcal{A}$. Policy π^e repeatedly samples trajectories of the form $a_1, o_1, r_1, a_2, o_2, r_2 \ldots$, starting from the initial system state (Algorithm 2). At the first iteration of the algorithm, before any model learning and policy optimization have occurred, π^e_1 is set to the initial behaviour policy π^e_{init} (line 2, Algorithm 1), a blind/open-loop policy such as a uniform random action policy. Thus, action sampling in line 6 of Algorithm 2 is not conditioned on the system state, $a_t \sim \pi^e_i(a)$, and the state update step in line 7 is skipped. In subsequent iterations, $i \geq 2$, π^e_i is updated based on π_{i-1}, the policy optimized in the previous iteration (line 8, Algorithm 1). To balance exploration and exploitation, possible choices for π^e include softmax and ϵ-greedy policies. In our experiments, we implemented the latter –

$$\pi^e_i(b, a) = \begin{cases} 1 - \epsilon & \text{for } a = \pi_{i-1}(b) \\ \frac{\epsilon}{|\mathcal{A}|-1} & \text{for all other actions } a \end{cases}$$

– where $\pi_{i-1}(b)$ is the optimal action at system state b, and ϵ is akin to the exploration constant.

The model learning and policy optimization steps are evoked every N_i sampled trajectories. We describe these steps in the next two sections.

Algorithm 2 Adaptive Goal-Directed Sampling

input: Behaviour policy π^e, model $\mathbf{M}$, number of trajectories to sample N, and maximum number of time steps T.
repeat
  $t \leftarrow 1$.
  Set b_t to the initial system state, m_0.
  repeat
    Sample $a_t \sim \pi_i^e(b_t, a)$. Execute a_t, get observation o_t and reward r_t from the environment.
    Update the system state b_{t+1} using equation (1) and model $\mathbf{M}$.
    $t \leftarrow t + 1$.
  until $t = T$.
until N trajectories have been sampled.
output: Sampled trajectories $\bar{\mathcal{G}}$.

4.2 Online Model Learning

Our model learning method is based on an online linear compression of the observed trajectories. In particular, we use an online singular value decomposition (SVD) method to update model parameters from streaming data.

To obtain a compact, low dimensional model representation, we learn a transformed PSR (TPSR) model [19]. TPSRs are a generalization of PSRs where the model parameters, m'_0, M'_{ao} and m'_{ao}, are transformed versions of PSR parameters, m_0, M_{ao} and m_{ao}, respectively, and give equivalent predictions for any tests τ. The TPSR model's compactness arises from its state being a linear combination of predictions for a potentially large set of PSR core tests. The state update proceeds as the PSR update in equation (1), with the transformed parameters substituting the PSR parameters.

Our model learning method relies on recent progress in learning TPSRs using spectral methods, as introduced in [2]. In this approach, instead of learning the system-dynamics matrix $\mathcal{D}$, only the thin SVD decomposition of $\mathcal{D}$ is maintained. New data is incorporated directly into the SVD decomposition using the online update method described in [5]. The computational complexity of the online update in the i-th iteration of our algorithm is $O(s_i \times |\mathbf{M}|^3)$, where $|\mathbf{M}|$ is the TPSR dimension, and s_i is the effective number of updated entries in $\mathcal{D}_i$ due to the additional data incorporated from newly sampled trajectories $\bar{\mathcal{G}}_i$. (In contrast, regular (batch) SVD has complexity $O(n^3)$, for $\mathcal{D}_i$ of size $n \times n$, i.e., the complexity is dependent on the size of $\mathcal{D}$, which could grow very quickly with the size of the action/observation spaces.) Further advantages of this model learning method are that it avoids actually building and storing $\mathcal{D}$, and it is amenable to tracking of non-stationary environments.

4.3 Policy Optimization

Our policy optimization method is based on fitted value iteration [9], an approach which formulates function approximation as a sequence of supervised learning

Algorithm 3 Fitted Q Iteration [7] for Predictive Models

Input: A set of tuples, $\mathcal{F}$, of the form (b_t, a_t, r_t, b_{t+1}), constructed from a set of sampled trajectories $\mathcal{G}$ and using a predictive model $\mathbf{M}$.
Initialization:
  $k \leftarrow 0$.
  Set $\hat{Q}_k(b, a)$ to be zero for all $b \in \mathcal{B}$, $a \in \mathcal{A}$.
Iterations:
repeat
  $k \leftarrow k + 1$.
  Build training set, $\mathcal{T} = \{(i^l, o^l), l = 1, \cdots, |\mathcal{F}|\}$, where:
    $i^l = (b_t^l, a_t^l)$, and, $o^l = r_t^l + \gamma \max_a \hat{Q}_{k-1}(b_{t+1}^l, a)$.
  Apply a regression algorithm to learn the function $\hat{Q}_k(b, a)$ from training set $\mathcal{T}$.
until stopping conditions are reached.
Output: policy π, where $\pi(b) \leftarrow \text{argmax}_a \hat{Q}(b, a)$.

problems. In particular, our algorithm performs something akin to batch mode Q-learning with function approximation, by applying an ensemble of trees algorithm to a sequence of regression problems. This fitted Q iteration approach has been shown to give good performance in fully observable systems and has good convergence properties [7]. We extend the algorithm to partially observable systems in Algorithm 3.

Given a set of trajectories $\mathcal{G}$ and a predictive model $\mathbf{M}$, the goal is to learn $\hat{Q}(b, a)$, an estimation of the expected cumulative reward as a result of executing action a from system state b. Given $\hat{Q}(b, a)$, the optimal action at b is $\pi(b) \leftarrow \text{argmax}_a \hat{Q}(b, a)$. The input to the algorithm is a set of tuples, (b_t, a_t, r_t, b_{t+1}), constructed from one-step system transitions in sampled trajectories of the form, $a_1, o_1, r_1, a_2, o_2, r_2, \cdots$. This requires tracking the system state b_t at every time step: for each trajectory, b_1 is set to the initial system state m_0, then for each subsequent time step, b_{t+1} is updated from b_t, a_t, and o_t, using equation (1). The set of tuples is used for solving a sequence of regression problems. At iteration k in the sequence, the regression function to be learned is the mapping from (b, a) to $\hat{Q}_k(b, a)$. The training set for fitting the regression function has as input the pairs (b_t, a_t), and target outputs $r_t + \gamma \max_a \hat{Q}_{k-1}(b_{t+1}, a)$, where $\hat{Q}_{k-1}(b, a)$ is the regression function learned in the previous iteration $k-1$, and γ is a discount factor. Repeated applications of the regression algorithm in successive iterations result in increasingly better approximations to the true Q-function.

We use extremely randomized trees (Extra-Trees)[8] as the regression algorithm. This predicts the value of a regression function using an ensemble of trees. The tree building procedure requires specification of the number of trees to build M_{tree}, and the minimal leaf size n_{min} (refer to [7] and [8] for more details).

5 Experimental Results

We evaluated our proposed algorithm on two Gridworld problems of different sizes, depicted in Figure 1. The agent does not sense its position directly, only

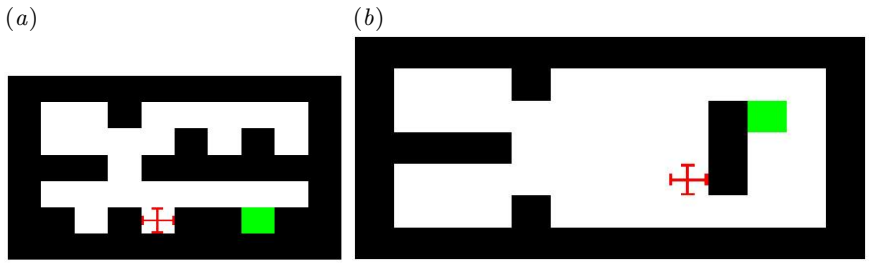

Fig. 1. Maps for Gridworld problems, GW-25 (a) and GW-47 (b). The cross is the starting position and the lightly shaded cell is the goal. Refer to the text for details.

the presence of adjacent walls in the four cardinal directions, hence there are 16 possible observations. In each time step, the agent takes a step in one of the four directions and the action succeeds with probability 0.8, otherwise, the agent moves in one of the perpendicular directions. The agent remains in place if it bumps into a wall. We refer to the two problems as GW-25 and GW-47, their respective environments contain 25 and 47 free cells. The agent receives zero reward everywhere except at the goal where it receives a reward of 1.

Our algorithm samples trajectories of fixed lengths (11 and 13 for GW-25 and GW-47, respectively). In the first iteration, 100 such trajectories are collected for GW-25 and 150 for GW-47, followed by 10 trajectories in each subsequent iteration. The trajectories are used to estimate probabilities of histories of up to length 6 and 8 for GW-25 and GW-47, respectively, and tests of up to length 4. The online TPSR learning process incorporates probability estimations from newly acquired data of each iteration and updates a 3-dimensional state-space TPSR model . We ran the fitted Q algorithm with 100 iterations and a discount factor of 0.95. The extremely randomized trees in the algorithm built single trees across all actions, in an ensemble consisting of 25 trees (M_{tree} parameter) and with minimum leaf size set at 15 data points (n_{min} parameter).

To illustrate the advantages of our approach we compare our algorithm with the usual (regular) approach in PSR literature which learns a model using a purely exploratory policy (uniform random action selection) for trajectory sampling, followed by planning only after model learning is completed [3].

Results of the policy performance with increasing number of sampled trajectories are presented in Figure 2. Plots (a), (b) and (c) compare the policies learned by our algorithm (adaptive) and the regular algorithm on GW-25. We show results for different settings of the exploration constant ϵ in our algorithm. Plot (d) shows performances on GW-47. Policies were evaluated by generating 100 trajectories and logging the proportion of trajectories that reached the goal.

Figure 2 shows that given the same number of sampled trajectories, our algorithm improves its policy faster than the usual PSR model learning and planning approach, and that the performance gap increases with increased problem size (compare plots (a) for GW-25 with (d) for GW-47.) Plots (a) to (c) show that

the difference between the results for $\epsilon = 0.2$ and $\epsilon = 0.3$ are minimal. However, for $\epsilon = 0.5$ two differences are noticeable. Firstly, for the initial 250 sampled trajectories, the policies for $\epsilon = 0.5$ improve slightly slower as compared to when ϵ is set to lower values. This is expected – with the bigger exploration factor, the optimal policy is invoked less. The second, more interesting observation is that, starting from the 350th sampled trajectory onwards, the performance of these policies continues to grow beyond the maximum achieved by the other policies. A possible explanation is that more exploration results in a more accurate model, which in turn is used to find a better policy. Yet, the majority of exploration is performed "on the way to the goal", leading to significantly better policies compared to those learned from completely random exploration. In a sense, it can be seen as a type of exploration-exploitation tradeoff which is different from the usual case in RL involving estimation of action values. Here, lack of exploration leads to inaccurate estimates of system dynamics, but exploring uniformly at random results in slow convergence to the optimal policy.

In Figure 3 we compare the execution times required for each algorithm to reach the same levels of performance on GW-25 (with $\epsilon = 0.5$ in our algorithm). Our algorithm essentially trades off data efficiency with computational complexity. Data efficiency is often crucial in real world settings – experience is more expensive than computation and the aim is to learn in as few trials or with as little experience as possible. Another important advantage is that our algorithm is 'anytime' – at any stage, the optimal policy (based on the most recently learned model) is immediately available to the agent. This is not the case for the usual approach of planning only after model learning is completed.

6 Related Work

There has been few prior work that address online learning of PSR models. McCracken and Bowling [15], and Aberdeen et al. [1], employ a constrained-gradient algorithm for online PSR learning. Boots and Gordon [2] learn TPSR models online by utilizing low-rank modifications of thin SVDs. However, none of these approaches address the problem of integrated learning and planning. In particular, although the work in [2] uses a similar model learning method as ours, it focuses on learning alone, while in [1], planning proceeds together with learning but the resultant policy from planning does not inform the behaviour policy for model learning.

The idea of not learning a complete model that can predict all possible futures has been explored in approximate PSRs ([6],[22],[24]), where the objective is to learn models that accurately predict only specific quantities of interest. The relevant predictions are specified beforehand and efficient use of sampled trajectories is achieved by aggregating tests and histories according to these predictions. In contrast, our approach achieves data efficiency by biasing sampling towards goal-directed trajectories, with sampling behaviour adaptively and automatically learned from system interaction.

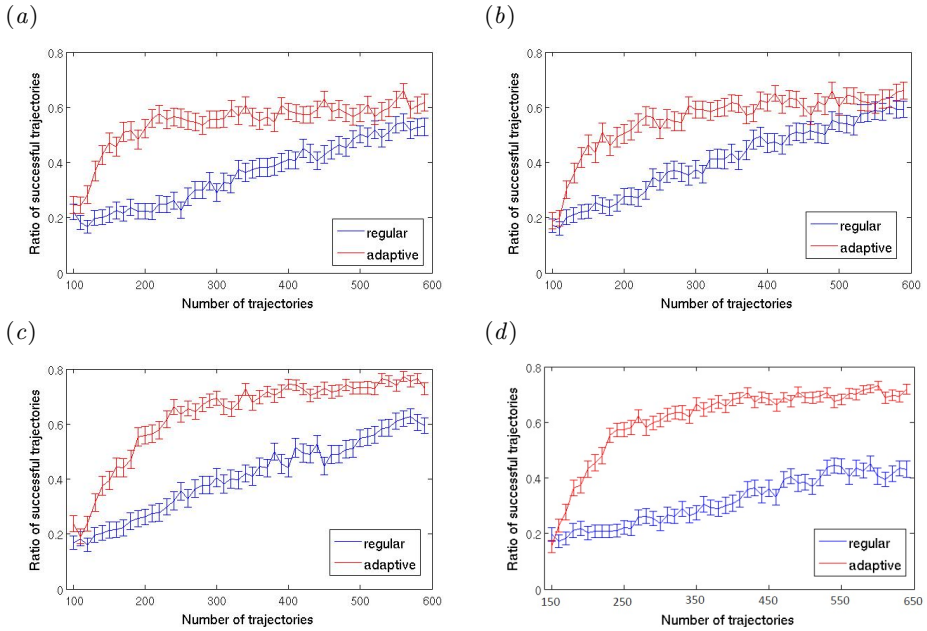

Fig. 2. Experimental results on the Gridworld problems depicted in Figure 1. Plots (a), (b) and (c) show policy performances on GW-25, with exploration constant ϵ set at 0.2, 0.3, and 0.5 respectively, in our algorithm. Plot (d) shows policy performances on GW-47, with $\epsilon = 0.2$ in our algorithm. All results were averaged over 100 runs. Error bars indicate standard deviations.

7 Discussion and Conclusion

This work proposes a new integrated learning and planning approach in PSRs. We show in preliminary experiments the advantages over doing learning and planning separately – given the same amount of data for learning, our method was able to learn a model and a value function that gives better policies, and the policy performance gap increases with increased problem size.

Our approach is suitable for domains where the initial system conditions are such that a good policy can be obtained without learning the dynamics of the whole state space or, equivalently, without searching in the whole space of policies. For example, in navigation tasks, the initial system state would be concentrated on a small portion of the environment if for example, the robot is learning how to navigate in a building, starting from a particular room within the building. Another necessary assumption is that the goal is reachable within the limits we impose on the trajectory length, starting from the initial system state. In practice, we make sure that the trajectories are of sufficient length in relation to the path to the goal.

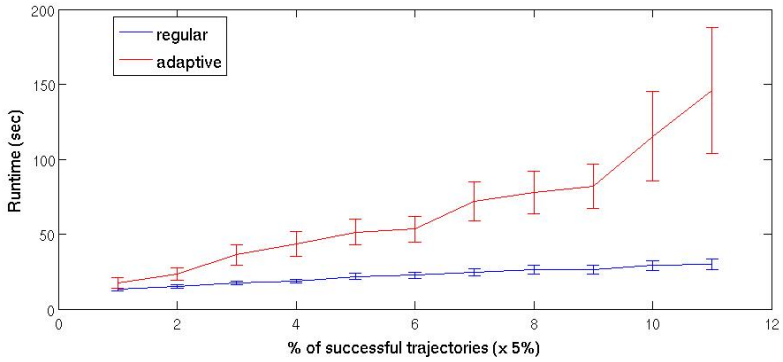

Fig. 3. Run time comparisons on Gridworld problem GW-25, with $\epsilon = 0.5$ in our algorithm. Results were averaged over 40 runs. Error bars indicate standard deviations. The experiments were carried on a PC with two 3.3GHz CPUs and 7GB of RAM, running Ubuntu 10.03 OS.

There are some interesting and important learning problems, which are particularly amenable to solution with our approach. In apprenticeship learning, the goal is to learn a policy that is at least as good or better than an expert. We can apply our method to apprenticeship learning in an unknown domain, by deriving the initial behaviour policy from the expert policy, and then iteratively improving upon it. Our method could also be applied to model customization. Given a basic model that is not customized to the current environment, our algorithm iteratively learns a better suited model. Lastly, given the online nature of our algorithm, and the integration of learning and planning, our approach can be extended to learning in non-stationary domains. We aim to investigate the applicability of our method to these classes of problems in our future work.

It would be interesting to compare empirically our algorithm with history-based approaches such as U-Tree [14], Monte-Carlo AIXI [25] and ΦMDPI [16]. These algorithms also integrate learning and planning, while attempting to balance exploration and exploitation. Their approach differs from ours in representing system state with histories rather than predicted probabilities of future experiences. We leave this also to future work.

Acknowledgments. The authors wish to thank Doina Precup (McGill University) for helpful discussions regarding this work. Funding was provided by the National Institutes of Health (grant R21 DA019800) and the NSERC Discovery Grant program.

References

1. Aberdeen, D., Buffet, O., Thomas, O.: Policy-gradients for PSRs and POMDPs. In: AISTATS (2007)

2. Boots, B., Gordon, G.J.: An online spectral learning algorithm for partially observable nonlinear dynamical systems. In: Proceedings AAAI (2011)
3. Boots, B., Siddiqi, S., Gordon, G.: Closing the learning-planning loop with predictive state representations. In: Proceedings of Robotics: Science and Systems (2010)
4. Bowling, M., McCracken, P., James, M., Neufeld, J., Wilkinson, D.: Learning predictive state representations using non-blind policies. In: Proceedings ICML (2006)
5. Brand, M.: Fast low-rank modifications of the thin singular value decomposition. Linear Algebra and its Applications 415, 20–30 (2006)
6. Dinculescu, M., Precup, D.: Approximate predictive representations of partially observable systems. In: Proceedings ICML (2010)
7. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning (2005)
8. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine Learning 63, 3–42 (2006)
9. Gordon, G.J.: Approximate Solutions to Markov Decision Processes. Ph.D. thesis, School of Computer Science, Carnegie Mellon University (1999)
10. Izadi, M., Precup, D.: Point-based planning for predictive state representations. In: Canadian Conference on AI (2008)
11. James, M.R., Wessling, T., Vlassis, N.: Improving approximate value iteration using memories and predictive state representations. In: AAAI (2006)
12. James, M.R., Singh, S., Littman, M.L.: Planning with predictive state representations. In: International Conference on Machine Learning and Applications. pp. 304–311 (2004)
13. Littman, M., Sutton, R., Singh, S.: Predictive representations of state. In: Advances in Neural Information Processing Systems (NIPS) (2002)
14. McCallum, A.K.: Reinforcement Learning with Selective Perception and Hidden State. Ph.D. thesis, University of Rochester (1996)
15. McCracken, P., Bowling, M.: Online discovery and learning of predictive state representations. In: Neural Information Processing Systems 18 (2006)
16. Nguyen, P., Sunehag, P., Hutter, M.: Feature reinforcement learning in practice. Tech. rep. (2011)
17. Poupart, P., Vlassis, N.: Model-based bayesian reinforcement learning in partially observable domains. In: Tenth International Symposium on Artificial Intelligence and Mathematics (ISAIM) (2008)
18. Rafols, E.J., Ring, M., Sutton, R., Tanner, B.: Using predictive representations to improve generalization in reinforcement learning. In: IJCAI (2005)
19. Rosencrantz, M., Gordon, G.J., Thrun, S.: Learning low dimensional predictive representations. In: Proceedings ICML (2004)
20. Ross, S., Pineau, J., Chaib-draa, B., Kreitmann, P.: A Bayesian approach for learning and planning in partially observable Markov decision processes. Journal of Machine Learning Research 12, 1655–1696 (2011)
21. Singh, S., James, M., Rudary, M.: Predictive state representations: A new theory for modeling dynamical systems. In: Proceedings UAI (2004)
22. Soni, V., Singh, S.: Abstraction in predictive state representations. In: AAAI (2007)
23. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press (1998)
24. Talvitie, E., Singh, S.: Simple local models for complex dynamical systems. In: Advances in Neural Information Processing Systems (NIPS) (2008)
25. Veness, J., Ng, K.S., Hutter, M., Uther, W., Silver, D.: A Monte-Carlo AIXI approximation. JAIR 40, 95–142 (2011)