Integrating Sample-Based Planning and Model-Based Reinforcement Learning

Thomas J. Walsh and **Sergiu Goschin** and **Michael L. Littman**
Department of Computer Science, Rutgers University
Piscataway, NJ 08854, USA
{thomaswa, sgoschin, mlittman}@cs.rutgers.edu

Abstract

Recent advancements in model-based reinforcement learning have shown that the dynamics of many structured domains (e.g. DBNs) can be learned with tractable sample complexity, despite their exponentially large state spaces. Unfortunately, these algorithms all require access to a planner that computes a near optimal policy, and while many traditional MDP algorithms make this guarantee, their computation time grows with the number of states. We show how to replace these over-matched planners with a class of sample-based planners—whose computation time is independent of the number of states—without sacrificing the sample-efficiency guarantees of the overall learning algorithms. To do so, we define sufficient criteria for a sample-based planner to be used in such a learning system and analyze two popular sample-based approaches from the literature. We also introduce our own sample-based planner, which combines the strategies from these algorithms and still meets the criteria for integration into our learning system. In doing so, we define the first complete RL solution for compactly represented (exponentially sized) state spaces with efficiently learnable dynamics that is both sample efficient and whose computation time does not grow rapidly with the number of states.

Introduction

Reinforcement-learning (Sutton and Barto 1998) or RL algorithms that rely on the most basic of representations (the so called "flat MDP") do not scale to environments with large state spaces, such as the exponentially sized spaces associated with factored-state MDPs (Boutilier, Dearden, and Goldszmidt 2000). However, model-based RL algorithms that use compact representations (such as DBNs), have been shown to have sample-efficiency (like PAC-MDP) guarantees in these otherwise overwhelming environments. Despite the promise of these sample-efficient learners, a significant obstacle remains in their practical implementation: they require access to a planner that guarantees ϵ-accurate decisions for the learned model. In small domains, traditional MDP planning algorithms like Value Iteration (Puterman 1994) can be used for this task, but for larger models, these planners become computationally intractable because they scale with the size of the state space. In a sense, the issue is not a learning problem—RL algorithms can learn the dynamics of such domains quickly. Instead, computation or *planning* is presently the bottleneck in model-based RL. This paper explores replacing the traditional flat MDP planners with *sample-based* planners. Such planners open up the planning bottleneck because their computation time is invariant with respect to the size of the state space.

We analyze the integration of sample-based planners with a class of model learners called KWIK learners, which have been instrumental in recent advances on the sample complexity of learning RL models (Li, Littman, and Walsh 2008). The KWIK framework and its agent algorithm, KWIK-Rmax, have unified the study of sample complexity across representations, including factored-state and relational models (Walsh et al. 2009). But, to date, only flat-MDP planners have been used with these KWIK learners.

Sample-based planners, starting with Sparse Sampling or SS (Kearns, Mansour, and Ng 2002), were developed to combat the curse of dimensionality in large state spaces. By conceding an exponential dependence on the horizon length, not finding a policy for every state, and randomizing the value calculations, SS attains a runtime that is independent of the number of states in a domain. The successor of SS, Upper Confidence for Trees (UCT), formed the backbone of some extremely successful AI systems for RTS games and Go (Balla and Fern 2009; Gelly and Silver 2007). Silver, Sutton, and Müller (2008) used a sample-based planner in a model-based reinforcement learning system, building models from experience and using the planner with this model. This architecture is similar to ours, but made no guarantees on sample or computational complexity, which we do in this work. We also note that while the literature sometimes refers to sample-based planners as "learning" a value function from rollouts, their behavior is better described as a form of *search* given a generative model.

The major contribution of this paper is to define the first complete RL solution for compactly represented (exponentially sized) state spaces with KWIK-learnable dynamics that is both sample efficient and whose computation time grows only in the size of the compact representation, not the number of states. To do so, we (1) describe a criteria for a general planner to be used in KWIK-Rmax and still ensure PAC-MDP behavior, (2) show that the original efficient sample-based planner, SS, satisfies these conditions, but the

more empirically successful UCT does not, and (3) introduce a new sample-based planning algorithm we call Forward Search Sparse Sampling (FSSS) that behaves more like (and sometimes provably better than) UCT, while satisfying the efficiency requirements.

Learning Compact Models

We first review results in RL under KWIK framework. We begin by describing the general RL setting.

Model-Based Reinforcement Learning and KWIK

An RL agent interacts with an environment that can be described as a Markov Decision Process (MDP) (Puterman 1994) $M = \langle S, A, R, T, \gamma \rangle$ with states S, actions A, a reward function $R : (S, A) \mapsto \Re$ with maximum reward $R_{\max}$, a transition function $T : (S, A) \mapsto Pr[S]$, and discount factor $0 \leq \gamma < 1$. The agent's goal is to maximize its expected discounted reward by executing an optimal policy $\pi^* : S \mapsto Pr[A]$, which has an associated value function $Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')$ where $V^*(s) = \max_a Q^*(s,a)$. In model-based RL, an agent initially does not know R and T. Instead, on each step it observes a sample from T and R, updating its model M', and then planning using M'. The KWIK framework, described below, helps measure the *sample complexity* of the model-learning component by bounding the number of inaccurate predictions M' can make.

Knows What It Knows or KWIK (Li, Littman, and Walsh 2008) is a framework developed for supervised active learning. Parameters ϵ and δ control the accuracy and certainty, respectively. At each timestep, an input x_t from a set X is presented to an agent, which must make a prediction $\hat{y}_t \in Y$ from a set of labels Y or predict $\perp$ ("I don't know"). If $\hat{y}_t = \perp$, the agent then sees a (potentially noisy) observation z_t of the true y_t. An agent is said to be *efficient* in the KWIK paradigm if with high probability $(1 - \delta)$: (1) The number of $\perp$ output by the agent is bounded by a polynomial function of the problem description; and, (2) Whenever the agent predicts $\hat{y}_t \neq \perp, ||\hat{y}_t - y_t|| < \epsilon$. The bound on the number of $\perp$ in the KWIK algorithm is related (with extra factors of $R_{\max}, (1-\gamma), \frac{1}{\epsilon}$, and δ) to the PAC-MDP (Strehl, Li, and Littman 2009) bound on the agent's behavior, defined as

Definition 1. *An algorithm $\mathcal{A}$ is considered **PAC-MDP** if for any MDP M, $\epsilon > 0$, $0 < \delta < 1$, $0 < \gamma < 1$, the sample complexity of $\mathcal{A}$, that is the number of steps t such that $V_t^{\mathcal{A}}(s_t) < V^*(s_t) - \epsilon$ is bounded by some function that is polynomial in the relevant quantities ($\frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{(1-\gamma)}$ and $|M|$) with probability at least $1 - \delta$.*

Here, $|M|$ measures the complexity of the MDP M based on the compact representation of T and R, not necessarily $|S|$ itself (e.g. in a DBN $|M|$ is $O(\log(|S|))$). To ensure this guarantee, a KWIK-learning agent must be able to connect its KWIK learner for the world's dynamics (KL) to a planner P that will interpret the learned model "optimistically". For instance, in small MDPs, the model could be explicitly constructed, with all $\perp$ predictions being replaced by transitions to a trapping $R_{\max}$ state. This idea is captured in the KWIK-Rmax algorithm (Li 2009).

Algorithm 1 KWIK-Rmax (Li 2009)

Agent knows S (in some compact form), A, γ, ϵ, δ
Agent has access to planner P guaranteeing ϵ accuracy
Agent has KWIK learner *KL* for the domain
$t = 0$, s_t = start state
while Episode not done **do**
  M' = KL with $\perp$ interpreted optimistically
  $a_t = P$.getAction(M', s_t)
  Execute a_t, view s_{t+1}, *KL*.update(s_t, a_t, s_{t+1}).
  $t = t + 1$

Intuitively, the agent uses KL for learning the dynamics and rewards of the environment, which are then given to an ϵ-optimal planner P. For polynomial $|S|$, a standard MDP planner is sufficient to guarantee PAC-MDP behavior. However, KWIK has shown that the sample complexity of learning representations of far larger state spaces can still be polynomially (in $|M|$) bounded (see below). In these settings, the remaining hurdle to truly efficient learning is a planner P that does not depend on the size of the state space, but still maintains the guarantees necessary for PAC-MDP behavior. Before describing such planners, we describe two domain classes with exponential $|S|$ that are still KWIK learnable.

Factored-state MDPs and Stochastic STRIPS

A factored-state MDP is an MDP where the states are comprised of a set of factors F that take on attribute values. While $|S|$ is $O(2^F)$, the transition function T is encoded using a Dynamic Bayesian Network or DBN (Boutilier, Dearden, and Goldszmidt 2000), where, for each factor f, the probability of its value at time $t+1$ is based on the value of its k "parent" factors at time t. Recently, KWIK algorithms for learning both the structure, parameters, and reward function of such a representation have been described (Li, Littman, and Walsh 2008; Walsh et al. 2009). Combining these algorithms, a KWIK bound of $\tilde{O}(\frac{F^{k+3}Ak}{\epsilon^4})$ can be derived—notice it is polynomial (for $k = \tilde{O}(1)$) in $F, A, \frac{1}{\epsilon}$ (and therefore $|M|$), instead of the (exponential in F) state space S. Similarly, a class of relational MDPs describable using *Stochastic STRIPS* rules has been shown to be partially KWIK learnable (Walsh et al. 2009). In this representation, states are described by relational fluents (e.g. Above(a,b)) and the state space is once again exponential in the domain parameters (the number of predicates and objects), but the preconditions and outcome probabilities for actions are KWIK learnable with polynomial sample complexity. We now present sample-based planners, whose computational efficiency do not depend on $|S|$.

Sample-based Planners

Sample-based planners are different from the original conception of planners for KWIK-Rmax in two ways. First, they require only a generative model of the environment, not access to the model's parameters. Nonetheless, they fit nicely with KWIK learners, which can be directly queried with state/action pairs to make generative predictions. Second, sample-based planners compute actions stochastically,

so their policies may assign non-zero probability to sub-optimal actions.

Formally, the MDP *planning problem* can be stated as, for state $s \in S$, select an action $a \in A$ such that over time the induced policy π will be ϵ-optimal, that is, $\sum_a \pi(s,a)Q^*(s,a) \geq V^*(s) - \epsilon$. Value Iteration (VI) (Puterman 1994) solves the planning problem by iteratively improving an estimate Q of Q^*, but takes time proportional to $|S|^2$ in the worst case. Instead, we seek a planner that fits the following criterion, adapted from Kearns, Mansour, and Ng (2002), that we later show is sufficient for preserving KWIK-Rmax's PAC-MDP guarantees.

Definition 2. *An* **efficient state-independent planner** P *is one that, given (possibly generative) access to an MDP model, returns an action a, such that the planning problem above is solved ϵ-optimally, and the algorithm's per-step runtime is independent of $|S|$, and scales no worse than exponentially in the other relevant quantities ($\frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma}$).*

Sample-based planners take a different view from VI. First, note that there is a horizon length H, a function of γ, ϵ and $R_{\max}$, such that taking the action that is near-optimal over the next H steps is still an ϵ-optimal action when considering the infinite sum of rewards (Kearns, Mansour, and Ng 2002). Next, note that the d-horizon value of taking action a from state s can be written $Q^d(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a'} Q^{d-1}(s',a')$, where $Q^1(s,a) = R(s,a)$. This computation can be visualized as taking place on a depth H tree with branching factor $|A||S|$. Instead of computing a policy over the entire state space, sample-based planners estimate the value $Q^H(s,a)$ for all actions a whenever they need to take an action from state s. Unfortunately, this insight alone does not provide sufficient help because the $|A||S|$ branching factor still depends linearly on $|S|$. But, the randomized Sparse Sampling algorithm (described below) eliminates this factor.

Sparse Sampling

The insight of Sparse Sampling or SS (Kearns, Mansour, and Ng 2002) is that the summation over states in the definition of $Q^H(s,a)$ need only be taken over a *sample* of next states and the size of this sample can depend on $R_{\max}, \gamma$, and ϵ instead of $|S|$. Let $K^d(s,a)$ be a sample of the necessary size C drawn according to the distribution $s' \sim T(s,a,\cdot)$. Then, the SS approximation can be written (for $s' \in K^d(s,a)$):

$$Q^d_{SS}(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^{d-1}_{SS}(s',a').$$

SS traverses the tree structure of state/horizon pairs in a bottom-up fashion. The estimate for a state at horizon t cannot be created until the values at $t+1$ has been calculated for all reachable states. It does not use intermediate results to focus computation on more important parts of the tree. As a result, its running time, both best and worst case, is $\Theta((|A|C)^H)$. Because it is state-space-size independent and solves the planning problem with ϵ accuracy (Kearns, Mansour, and Ng 2002), it satisfies Definition 2, making it an attractive candidate for integration into KWIK-Rmax. We prove this combination preserves PAC-MDP behavior later. Unfortunately in practice, SS is typically quite slow because its search of the tree is not focused, a problem addressed by its successor, UCT.

Upper Confidence for Tree Search

Conceptually, UCT (Kocsis and Szepesvári 2006) takes a top-down approach (from root to leaf), guided by a non-stationary *search policy*. In top-down approaches, planning proceeds in a series of *trials*. In each trial (Algorithm 2), a search policy is selected and followed from root to leaf and the state-action-reward sequence is used to update the Q estimates in the tree. Note that, if the search policy is the optimal policy, value updates can be performed via simple averaging and estimates converge quickly. Thus, in the best case, a top-down sample-based planner can achieve a running time closer to C, rather than $(|A|C)^H$.

In UCT, the sampling of actions at a state/depth node is determined by $v + \max_a \sqrt{2\log(n_{sd})/n_a}$, where v is the average value of action a from this state/depth pair based on previous trials, n_{sd} counts the number of times state s has been visited at depth d and n_a is the number of times a was tried there. This quantity represents the upper tail of the confidence interval for the node's value, and the strategy encourages an aggressive search policy that explores until it finds fairly good rewards, and then only periodically explores for better values. While UCT has performed remarkably in some very difficult domains, we now show that its theoretical properties do not satisfy the efficiency conditions in Definition 2.

Algorithm 2 UCT(s, d) (Kocsis and Szepesvári 2006)

if $d = 1$ **then**
  $Q^d(s,a) = R(s,a), \forall a$
else
  $a^* = \text{argmax}_a(Q^d(s,a) + \sqrt{2\log(n_{sd})/n_{a^*sd}})$
  $s' \sim T(s,a,\cdot)$
  $v = R(s,a^*) + \gamma$ UCT(s', $d-1$)
  $n_{sd} = n_{sd} + 1$
  $n_{a^*sd} = n_{a^*sd} + 1$
  $Q^d(s,a^*) = (Q^d(s,a^*) \times (n_{a^*sd} - 1) + v)/n_{a^*sd}$
  return v

A Negative Case for UCT's Runtime

UCT gets some of its advantages from being aggressive. Specifically, if an action appears to lead to low reward, UCT quickly rules it out. Unfortunately, sometimes this conclusion is premature and it takes a very long time for UCT's confidence intervals to grow and encourage further search. A concrete example shows how UCT can take super-exponential trials to find the best action from s.

The environment in Figure 1 is adapted from Coquelin and Munos (2007) with the difference that it has only a polynomial number of states in the horizon. The domain is deterministic, with two actions available at each non-goal state. Action a_1 leads from a state s_i to state s_{i+1} and yields 0 reward, except for the transition from state s_{D-1} to the goal state s_D where a reward of 1 is received. Action a_2 leads

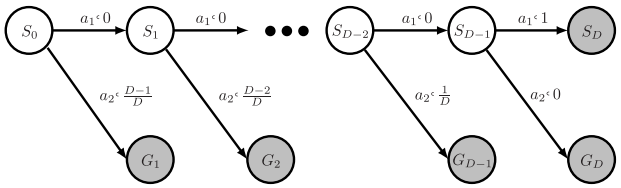

Figure 1: A simple domain where UCT can require super-exponential computation to find an ϵ-optimal policy.

from a state s_{i-1} to a goal state G_i while receiving a reward of $\frac{D-i}{D}$. The optimal action is to take a_1 from state s_0.

Coquelin and Munos (2007) proved that the initial number of timesteps needed to reach node s_D (which is always optimal in their work) for the first time is $\Omega(exp(exp(\overbrace{\cdots}^{D-4}(exp(2)))$. Given this fact, we can state

Proposition 1. *For the MDP in Figure 1, for any $\epsilon < \frac{1}{D}$, the minimum number of trials (and therefore the computation time) UCT needs before node s_D is reached for the first time (and implicitly the number of steps needed to ensure the behavior is ϵ-optimal) is a composition of $O(D)$ exponentials. Therefore, UCT does not satisfy Definition 2.*

Proof. Assume action a_2 is always chosen first when a previously unknown node is reached[1]. Let $\epsilon < \frac{1}{D}$. According to the analysis of Coquelin and Munos (2007), action a_2 is chosen $\Omega(exp(exp(\cdots(exp(2)))$ times before s_D is reached the first time. The difference between the values of the two actions is at least $\frac{1}{D} > \epsilon$, implying that, with probability 0.5, if UCT is stopped before the minimum number of necessary steps to reach s_D, it will return a policy (and implicitly an action) that is not ϵ-optimal. □

We note that this result is slightly different from the regret bound of Coquelin and Munos (2007), which is not directly applicable here. Still, the empirical success of UCT in many real world domains makes the use of a top-down search technique appealing. In the next section, we introduce a more conservative top-down sample-based planner.

Forward Search Sparse Sampling

Our new algorithm, Forward Search Sparse Sampling (FSSS) employs a UCT-like search strategy without sacrificing the guarantees of SS. Recall that $Q^d(s,a)$ is an estimate of the depth d value for state s and action a. We introduce upper and lower bounds $U^d(s)$ and $L^d(s)$ for states and $U^d(s,a)$ and $L^d(s,a)$ for state–action pairs.

Like UCT, it proceeds in a series of top-down trials (Algorithm 3), each of which begins with the current state s and depth H and proceeds down the tree to improve the estimate of the actions at the root. Like SS, it limits its branching factor to C. Ultimately, it computes precisely the same value as SS, given the same samples. However, it benefits from a kind of pruning to reduce the amount of computation needed in many cases.

[1]Similar results can be obtained if ties are broken randomly.

Algorithm 3 FSSS(s, d)

if $d = 1$ (leaf) **then**
  $L^d(s,a) = U^d(s,a) = R(s,a), \forall a$
  $L^d(s) = U^d(s) = \max_a R(s,a)$
else if $n_{sd} = 0$ **then**
  for each $a \in A$ **do**
    $L^d(s,a) = V_{\min}$
    $U^d(s,a) = V_{\max}$
    for C times **do**
      $s' \sim T(s,a,\cdot)$
      $L^{d-1}(s') = V_{\min}$
      $U^{d-1}(s') = V_{\max}$
      $K^d(s,a) = K^d(s,a) \cup \{s'\}$
$a^* = \text{argmax}_a\, U^d(s,a)$
$s^* = \max_{s' \in K^d(s,a^*)}(U^{d-1}(s') - L^{d-1}(s'))$
FSSS(s^*, $d-1$)
$n_{sd} = n_{sd} + 1$
$L^d(s,a^*) = R(s,a^*) + \gamma \sum_{s' \in K^d(s,a^*)} L^{d-1}(s')/C$
$U^d(s,a^*) = R(s,a^*) + \gamma \sum_{s' \in K^d(s,a^*)} U^{d-1}(s')/C$
$L^d(s) = \max_a L^d(s,a)$
$U^d(s) = \max_a U^d(s,a)$

When $L^H(s,a^*) \geq \max_{a \neq a^*} U^H(s,a)$ for $a^* = \text{argmax}_a\, U^H(s,a)$, no more trials are needed and a^* is the best action at the root. The following propositions show that, unlike UCT, FSSS solves the planning problem in accordance with Definition 2.

Proposition 2. *On termination, the action chosen by FSSS is that same as that chosen by SS.*

Using the definition of $Q^d(s,a)$ in SS, it is straightforward to show that $L^d(s,a)$ and $U^d(s,a)$ are lower and upper bounds on its value. If the termination condition is achieved, these bounds indicate that a^* is the best.

Proposition 3. *The total number of trials of FSSS before termination is bounded by the number of leaves in the tree.*

Note that each trial ends at a leaf node. We say a node s at depth d is *closed* if its upper and lower bounds match, $L^d(s) = U^d(s)$. A leaf is closed the first time it is visited by a trial. We argue that every trial closes a leaf.

If the search is not complete, the root must not be closed. Now, inductively, assume the trial has reached a node s and depth d that is not closed. For the selected action a^*, it follows that $L^d(s,a^*) \neq U^d(s,a^*)$ (or s must be closed). That means there must be some $s' \in K^d(s,a^*)$ such that s' at $d-1$ is not closed, otherwise the upper and lower bound averages would be identical. FSSS chooses the s' with the widest bound. Since each step of the trial leads to a node that is not closed, it must terminate at a leaf that had not already been visited. Once the leaf is visited, it is closed and cannot be visited again.

Another property of FSSS is that it implements a version of classical search-tree pruning. There are several kinds of pruning suggested for SS (Kearns, Mansour, and Ng 2002), but they all boil down to: Don't do evaluations in parts of the tree where the value (even if it is maximized) cannot exceed

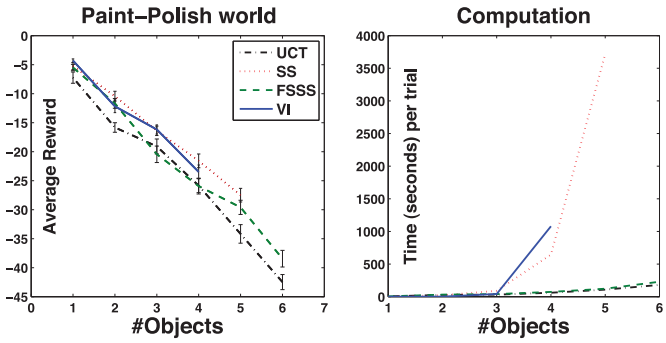

Figure 2: Planners in Paint-Polish world with increasing objects (40 runs). The optimal policy's average reward (VI) decreases linearly with the number of objects. Note VI and SS become intractable, as seen in the computation time plot.

the value of some other action (even if it is minimized). Our choice of taking the action with the maximum upper bound achieves this behavior. One advantage FSSS has over classical pruning is that our approach can interrupt the evaluation of one part of the tree if partial information indicates it is no longer very promising.

Also, as written, a trial takes time $O(H(|A|+C))$ because of the various maximizations and averages. With more careful data structures, such as heaps, it can be brought down to $O(H(\log|A| + \log C))$ per trial. Also, if better bounds are available for nodes of the tree, say from an admissible shaping function, L and U can be initialized accordingly.

Figure 2 shows the three sample-based planners discussed above performing in the "Paint-Polish" relational MDP (Stochastic STRIPS) from Walsh et al. (2009). The domain consists of $|O|$ objects that need to be painted, polished, and finished, but the stochastic actions sometimes damage the objects and need to be repeated. Like most relational domains $|S|$ is exponential in $|O|$ and here $|A|$ grows linearly in $|O|$. With increasing $|O|$, VI quickly becomes intractable and SS falters soon after because of its exhaustive search. But, UCT and FSSS can still provide passable policies with low computational overhead. FSSS's plans also remain slightly more consistent than UCT, staying closer to the linearly decreasing expected reward of π^* for increasing O. For both of those planners 2000 rollouts were used to plan at each step.

Learning with Sample-Based Planners

We now complete the connection of sample-based planners with the KWIK-Rmax algorithm by proving that the PAC-MDP conditions are still satisfied by our modified KWIK-Rmax algorithm. Note this integration requires the planner to see an optimistic interpretation of the learned model. The needed modification for an algorithm like Value Iteration would be to replace any unknown transitions ($\perp$) with a high-valued "$R_{\max}$" state. In our sample-based algorithms, we will use $\frac{R_{\max}}{1-\gamma}$ as the value of any (s,d,a) triple in the tree where the learner makes such an unknown prediction. Note the new algorithm no longer has to explicitly build M', and can instead use KL directly as a generative model for P. Thus modified, the algorithm has the following property.

Theorem 1. *The KWIK-Rmax algorithm (Algorithm 1) using a sample-based planner P that satisfies Definition 2 and querying KL as a generative model with the "optimism" modification described above is PAC-MDP (Definition 1) and has computation that grows only with $|M|$, not $|S|$.*

Proof Sketch of Theorem 1

The full proof is similar to the original KWIK-Rmax proof (Li 2009), so we describe only the lemmas that must be adapted due to the use of sample-based planners.

The crux of the proof is showing KWIK-Rmax with sample-based planners that satisfy Definition 2 satisfies the 3 sufficient conditions for an algorithm to be PAC-MDP: *optimism*, *accuracy* and *bounded number of "surprises"*. An *optimistic* model is one for which the estimated value function in a given state is greater than the optimal value function in the real MDP. A model is *accurate* when the estimated value function is close enough to the value of the current policy in the known part of the MDP. Since our new algorithm does not explicitly build this MDP (and instead connects KL and P directly), these two conditions are changed so that at any timestep the estimated value of the optimal stochastic policy in the current estimated MDP (from KL) is optimistic and accurate. A *surprise* is a discovery event that changes the learned MDP (in KL).

The following lemma states that the algorithm's estimated value function is optimistic for any time step t.

Lemma 1. *With probability at least $1-\delta$, $V_t^{\pi_t}(s) \geq V^*(s) - \epsilon$ for all t and (s,a), where $\pi(t)$ is the policy returned by the planner.*

The proof is identical to the original proof of Li (2009) (see Lemma 34) with the extra observation that the planner computes an implicit $V_t^{\pi_t}$ function of a stochastic policy.

Now, turning to the accuracy criterion, we can use a variation of the Simulation Lemma (c.f. Lemma 12 of Strehl, Li, and Littman (2009)) that applies to stochastic policies, and bounds the difference between the value functions of a policy in two MDPs that are similar in terms of transitions and rewards. The intuition behind the proof of this new version is that the stationary stochastic policy π in MDPs M_1 and M_2 induces two Markov chains M_1' and M_2' with transitions $T_1'(s,s') = \sum_a \pi(s,a)T_1(s,a,s')$ and rewards $R_1'(s) = \sum_a \pi(s,a)R(s,a)$ (analogously for T_2', R_2'). By standard techniques, we can show these transition and reward functions are close.

Because the two Markov chains are ϵ-close, they have ϵ-close value functions and thus, the value functions of π in MDPs M_1 and M_2 are bounded by the same difference as between the optimal value functions in MDPs M_1' and M_2'. According to the standard simulation lemma, the difference is $\frac{\epsilon_R+\gamma V_{max}\epsilon_T}{1-\gamma}$. From there, the following lemma bounds the accuracy of the policy computed by the planner:

Lemma 2. *With probability at least $1-\delta$, $V_t^{\pi_t}(s_t) - V_{M_K}^{\pi_t}(s_t) \leq \epsilon$, where $\pi(t)$ is the policy returned by the planner, and M_K is the known MDP.*

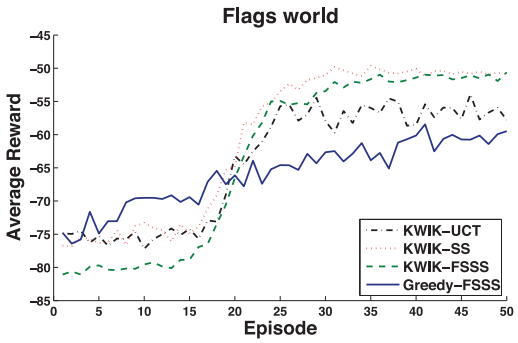

Figure 3: KWIK and an ϵ-greedy learner in the 5×5 Flags Domain (30 runs each) with 6 flags possibly appearing.

The third condition (bounded number of surprises) follows trivially from the KWIK bound for KL. With these new lemmas, the rest of the proof of Theorem 1 follows along the lines of the original KWIK-Rmax proof (Li 2009). Thus, KWIK-Rmax with an ϵ-accurate sample-based planner satisfying Definition 2 (like SS or FSSS) is PAC-MDP.

The full learning algorithm is empirically demonstrated in Figure 3 for a "Flags" factored MDP. The agent is in a $n \times n$ grid with slippery transitions where flags can appear with different probabilities in each cell, giving us $|S| = O(n^2 2^{n^2})$. Step costs are -1 except when the agent captures a flag (0) and flags do not appear when the agent executes the capture-flag action. In our experiment, only 6 cells actually have flags appear in them, 3 in the top-right (probability 0.9) and 3 in the bottom-left (0.02), though the agent does not know this a priori. Instead, the agent must learn the parameters of the corresponding DBN (though it is given the structure). The SS and FSSS KWIK agents find a near-optimal policy while the inferior exploration of ϵ-greedy (the exploration strategy used in previous work on sample-based planners in model-based RL) falls into a local optimum. The difference between the ϵ-greedy learner and the two best KWIK learners is statistically significant. Our implementation of VI could not handle the state space.

Related Work and Conclusions

Other researchers have combined learned action-operators with sample-based planning. Silver, Sutton, and Müller (2008) used a version of the model-based RL algorithm Dyna with UCT to play Go. The exploration strategy in this work was ϵ-greedy, which is known to have negative sample-complexity results compared to the Rmax family of algorithms (Strehl, Li, and Littman 2009). In relational MDPs, SS, has been used in a similar manner (Pasula, Zettlemoyer, and Kaelbling 2007; Croonenborghs et al. 2007). However, the first of these papers dealt with logged data, and the second used a heuristic exploration strategy with no sample efficiency guarantees. In contrast to all of these, we have presented a system that utilizes sample-based planning in concert with sample-efficient learners. We have thus presented the first end-to-end system for model-based reinforcement learning in domains with large state spaces where neither its sample nor computational efficiency is made intractable by the exponential number of states.

One area of future work is investigating model-based RL's integration with planners for specific domain classes. These include planners for DBNs (Boutilier, Dearden, and Goldszmidt 2000) and Stochastic STRIPS operators (Lang and Toussaint 2009). The former has already been integrated into a learning system (Degris, Sigaud, and Wuillemin 2006), but without analysis of the resulting complexity.

Acknowledgments This work was partially supported by NSF RI-0713148.

References

Balla, R.-K., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *IJCAI*.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1):49–107.

Coquelin, P.-A., and Munos, R. 2007. Bandit algorithms for tree search. In *UAI*.

Croonenborghs, T.; Ramon, J.; Blockeel, H.; and Bruynooghe, M. 2007. Online learning and exploiting relational models in reinforcement learning. In *IJCAI*.

Degris, T.; Sigaud, O.; and Wuillemin, P.-H. 2006. Learning the structure of factored Markov decision processes in reinforcement learning problems. In *ICML*.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *ICML*.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning* 49:193–208.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *ECML*.

Lang, T., and Toussaint, M. 2009. Approximate inference for planning in stochastic relational worlds. In *ICML*.

Li, L.; Littman, M. L.; and Walsh, T. J. 2008. Knows what it knows: A framework for self-aware learning. In *ICML*.

Li, L. 2009. *A Unifying Framework for Computational Reinforcement Learning Theory*. Ph.D. Dissertation, Rutgers University, NJ, USA.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29:309–352.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley.

Silver, D.; Sutton, R. S.; and Müller, M. 2008. Sample-based learning and search with permanent and transient memories. In *ICML*.

Strehl, A. L.; Li, L.; and Littman, M. L. 2009. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research* 10(2):413–444.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Walsh, T. J.; Szita, I.; Diuk, C.; and Littman, M. L. 2009. Exploring compact reinforcement-learning representations with linear regression. In *UAI*.