Improving Approximate Value Iteration using Memories and Predictive State Representations

Michael R. James
AI and Robotics Group, Technical Research Dept.
Toyota Technical Center
michael.r.james@gmail.com

Ton Wessling and **Nikos Vlassis**
Intelligent Autonomous Systems Group
Informatics Institute, University of Amsterdam
{twesslin, vlassis}@science.uva.nl

Abstract

Planning in partially-observable dynamical systems is a challenging problem, and recent developments in point-based techniques such as Perseus significantly improve performance as compared to exact techniques. In this paper, we show how to apply these techniques to new models for non-Markovian dynamical systems called Predictive State Representations (PSRs) and Memory-PSRs (mPSRs). PSRs and mPSRs are models of non-Markovian decision processes that differ from latent-variable models (e.g. HMMs, POMDPs) by representing state using only observable quantities. Further, mPSRs explicitly represent certain structural properties of the dynamical system that are also relevant to planning. We show how planning techniques can be adapted to leverage this structure to improve performance both in terms of execution time as well as quality of the resulting policy.

Introduction

Value-iteration algorithms for partially-observable dynamical systems compute, or approximate, a value function with respect to the *state representation* used. For instance, in POMDPs, the value function will be over belief-states. However, recent work (Littman, Sutton, & Singh, 2001; Singh, James, & Rudary, 2004) has begun to question whether other types of state representations would be advantageous in some situations. Two types of models that have emerged are predictive state representations (PSRs) (Littman, Sutton, & Singh, 2001), and memory-PSRs (mPSRs) (James, Singh, & Wolfe, 2005).

Memory-PSRs are of special interest, and use a memory of the past as a part of their state representation. In this paper, we extend the work in James & Singh (2005), and show how memories can be incorporated into value-iteration algorithms, including approximate point-based algorithms such as PERSEUS (Spaan & Vlassis, 2005). We demonstrate how the use of memories can make planning more efficient, and present empirical results that support this.

Along the way, we demonstrate how to adapt point-based value-iteration algorithms to PSRs. It turns out that point-based value iteration for PSRs and for POMDPs are very similar, with only minor changes needed. Planning in the traditional sense (with an a priori known model) would not see much advantage in using PSRs over POMDPs (although mPSRs would have advantages), but planning in somewhat unknown environments would benefit from using PSRs, as it has been demonstrated (Wolfe, James, & Singh, 2005) that PSRs (and mPSRs) are potentially more easily learned than POMDPs, so one could imagine interleaving model learning with planning (as it was recently done in POMDPs (Shani, Brafman, & Shimony, 2005)).

Although planning with PSRs may have advantages for certain situations, the major contribution of this paper is to demonstrate how the use of memory in point-based value iteration algorithms can lead to significant benefits.

This paper will first briefly review relevant material, including PERSEUS, PSRs, and mPSRs. Then the adaptation of point-based value-iteration algorithms to PSRs and mPSRs is presented, followed by empirical results.

Background material

Before presenting the details of approximate planning with PSRs and mPSRs, first some notation and related material need be presented. Consider a discrete-time, finite-space dynamical system in which actions are chosen from a set $\mathcal{A}$, observations from a set $\mathcal{O}$, rewards are assumed to take on a finite number of values and are treated as an additional dimension of the observation. There is exactly one action and one observation per time-step. There have been many algorithms developed to calculate optimal action choice, with respect to the long-term discounted reward, where γ is the discount factor. For partially-observable dynamical systems, the current state-of-the-art algorithms are point-based value iteration algorithms, applied to models called partially observable Markov decision processes (POMDPs).

POMDPs are models of dynamical systems based on underlying latent variables called nominal-states. In POMDPs, the current state—called the belief-state—is represented as a probability distribution over nominal-states. However, recent developments in modeling of dynamical systems are concerned with building models that are not dependent on any latent variables. Examples of these new types of models include observable operator models (OOMs) (Jaeger, 2000), predictive state representations (PSRs) , memory-PSRs (mPSRs), and TD-networks (Sutton & Tanner, 2004). In this paper, we will make use of PSRs and mPSRs.

POMDPs

POMDPs are well-documented in the literature, so we omit a full definition here in favor of a brief overview (see (Spaan & Vlassis, 2005) among many others for a full definition). POMDP models are defined in using probabilities of transitions between nominal-states ($pr(s'|s,a)$), probabilities of observations given nominal-states ($pr(o|s,a)$), and expected reward given nominal-states (expressed as a vector r_a with one entry per nominal-state), where s and s' are nominal-states.

PERSEUS: Randomized Point-based Value Iteration

Approximate, point-based planning algorithms, such as PERSEUS , for partially observable dynamical systems use an approximation of the value function to plan with. These algorithms rely on the fact that updating the value function for one point will also improve the value-function approximation for nearby points. The basic idea is to calculate the value function (and its gradient) only at a small number of points, and the value function for the other points will be "carried along" using this property. This results in an approximation of the value function using only a fraction of the computational requirements.

The PERSEUS algorithm uses randomized selection of points in the process of updating the value function. There are two main sets of vectors used in the PERSEUS algorithm: the set P of points, each of which is a belief-state vector; and the set S_n which represents the approximate value function V_n at step n. The value function is a piecewise linear convex function over belief states, defined as the upper surface of the vectors $\alpha \in S_n$. Therefore, the value for belief state b is given by

$$V_n(b) = \max_{\alpha \in S_n} b^T \alpha \tag{1}$$

To compute the value of an $n+1$ step value function at a point b, and given an n step value function, the so-called backup operator is used.

$$backup(b) = \operatorname*{argmax}_{a \in \mathcal{A}} b^T r_a + \gamma \sum_o \operatorname*{argmax}_{\alpha \in S_n} b^T \sum_{s'} pr(o|s',a) pr(s'|s,a) \alpha(s') \tag{2}$$

Note that there is some abuse of notation because argmax is used to return the corresponding vectors, and so the backup operator actually returns a vector α. For POMDPs, the basic outline of PERSEUS is:

1. Randomly collect a set P of reachable points (reachable when following a random policy from a starting belief b_0), each of which is a belief-state vector, set $n = 0$, and initialize S_0 to contain a single α vector with minimal values.
2. Set $S_{n+1} = \emptyset$, and initialize the temporary set $\overline{P}$ to P, where $\overline{P}$ is the set of non-improved points.
3. Sample a belief-state b uniformly at random from $\overline{P}$, and compute $\alpha = backup(b)$.
4. If $b^T\alpha \geq V_n(b)$ then add α to S_{n+1}, otherwise add $\alpha' = \operatorname{argmax}_{\alpha'' \in S_n} b^T \alpha''$ to S_{n+1}.
5. Compute $\overline{P} = \{b \in \overline{P} : V_{n+1}(b) - \epsilon < V_n(b)\}$. If $\overline{B}$ is empty, set $S_n = S_{n+1}$ and go to step 2, otherwise go to step 3.

The algorithm will stop according to some condition, typically either a condition on the value function or on the number of iterations. Approximate point-based value function methods such as PERSEUS are currently the fastest planning algorithms for partially-observable dynamical systems.

Predictive-State Models

There has been recent interest in models of dynamical systems that do away with latent, unobservable variables, and express the *state* of the dynamical system only in terms of observable quantities. Two such models are predictive state representations (PSRs) and memory-PSRs (mPSRs). In PSRs, the state of the dynamical systems is expressed in terms of possible future events, e.g. what is the probability that this window will break if I throw this rock at it? A set of such probabilities is referred to as predictive state. For many classes of dynamical systems, including all POMDPs, it has been shown that special sets of such future events, called the set of *core tests* can fully represent the state, just as state representations using latent-variables, such as POMDPs, can.

The major advantage of using only observable quantities to represent state is that the model is then grounded in quantities that can be measured and confirmed by observing the dynamical system itself. Thus, tasks like learning of models from observed data become much easier.

The idea behind memory-PSRs is a memory of past (observable) events can be used in conjunction with predictions in the state representation. It has been shown that simple types of memory, such as the most recent observation, may be advantageously included in the state representation. The resulting mPSR will contain both memories of the past as well as predictions about the future in its state representation. Further, using memories can result in a more compact model, and have advantages for planning algorithms. This will be discussed in detail later in the paper.

PSRs Consider a discrete-time dynamical system as defined above. At time s, the *history* h_s is the sequence $a^1o^1...a^so^s$. For any history h, the dynamical system defines the probability $pr(o|ha)$ of seeing observation o given that action a is taken. This is the *prediction* $p(t|h) = pr(o|ha)$ of the one-step *test* $t_1 = ao$ at history h.

More generally, a test $t = a_1o_1, ...a_ko_k$ is a sequence of alternating actions ($a_i \in \mathcal{A}$) and observations ($o_j \in \mathcal{O}$), and the prediction of a test t at history h is $p(t|h) = pr(o_1, ...o_k|ha_1, ...a_k)$, i.e., the conditional probability that the observation sequence occurs, given that the action sequence is taken after history h [1].

[1]It has been recently pointed out that this definition relies on the fact that the action sequence is "blind" or open-loop, or equivalently, that the random variables a_i do not correspond to the actions that are actually taken, but instead correspond to the actions that are designated to be taken if the test were to be executed.

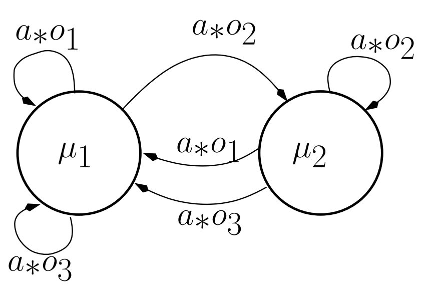

Figure 1: A depiction of memories and the deterministic transitions between them based on actions and observations. In this case, the memories are based on the most recent observation and actions are irrelevant: μ_1 corresponds to observations o_1 and o_3, and μ_2 corresponds to observation o_2.

A basic part of a PSR model is the set of *core tests* Q, the predictions of which are maintained in the size $(|Q| \times 1)$ *prediction vector* $p(Q|h)$, which is the PSR's state representation. The minimum number n of core tests required to maintain a sufficient statistic of state is called the dimension of the dynamical system, and for this paper we choose Q so that $|Q| = n$.

The prediction vector can be used to calculate the predictions of any other tests, and can be kept updated as actions are taken and observations occur. A PSR model defines a set of *model parameters* that perform both these tasks. The model parameters consist of a set of $(n \times n)$ matrices M_{ao}, and $(n \times 1)$ vectors m_{ao}, for all a, o. To predict the test $t = a_1o_1, ...a_ko_k$ at history h, the computation is

$$p(t|h) = p(Q|h)^T M_{a_1o_1}...M_{a_{k-1}o_{k-1}} m_{a_ko_k} \quad (3)$$

and to update state from history h as action a is taken and o is observed, the new prediction vector is

$$p(Q|hao) = \frac{p(Q|h)^T M_{ao}}{p(Q|h)^T m_{ao}} \quad (4)$$

The immediate expected reward for action a at any history h, denoted $R(h, a)$, is computed as

$$\begin{aligned} R(h,a) &= \sum_r r \cdot pr(r|h,a) \\ &= \sum_r r \sum_{o \in \mathcal{O}^r} p(Q|h)^T m_{ao} \\ &= p(Q|h)^T \sum_r \sum_{o \in \mathcal{O}^r} r \cdot m_{ao}. \quad (5) \end{aligned}$$

where $\mathcal{O}^r$ is the set of observations in which the reward component takes on value r. Thus, we can define a $(n \times 1)$ vector r^a as

$$r^a = \sum_r \sum_{o \in \mathcal{O}^r} r \cdot m_{ao} \quad (6)$$

such that $R(h, a) = p(Q|h)^T r^a$. The set of vectors r^a for all $a \in \mathcal{A}$ are used as a part of the PSR model when planning.

mPSRs Memories, in this context, are typically thought of as some portion of the observable past, and a memory-PSR defines state to be a pair containing the memory and a prediction vector. A simple, and often useful, example is memory containing the most recent observation. As new observations occur, the memory portion of state is updated along with the prediction vector. In many dynamical systems, adding memory to state will make the overall state representation smaller, and can also result in a smaller model of the dynamical system, i.e. the mPSR model has fewer model parameters than the PSR model. Intuitively, the memory contains additional information about the state, so that fewer predictions are needed, reducing the overall model size.

We will define memories to be more general than just a most-recent subsequence, but in practice it turns out that using a most-recent subsequence of history is both conceptually simple and gives good results. This is due to the fact that the most recent actions and observations often give the most information about the current state. We now turn to the definition of mPSRs.

Let $\mu_1 \ldots \mu_m$ represent the m distinct memories. Each memory will have an associated set of μ-core tests $Q^{\mu_1} \ldots Q^{\mu_m}$ respectively. Let the memory at history h be denoted $\mu(h)$. Then the mPSR state at history h is the pair $[\mu(h), p(Q^{\mu(h)}|h)]$. The number of μ-core tests may be different for each memory, and can never be larger—but can be much smaller—than the number of PSR core tests for the same system. Another relation is that the sum of μ-core tests for all memories must be $\geq$ than the number of PSR core tests.

For each memory, μ_i, there is an associated set of update matrices $M^{\mu_i}_{ao}$ and vectors $m^{\mu_i}_{ao}$ for all a, o. This part of the mPSR definition imposes the first limitation on memories, which is that, given a memory μ, action a, and observation o, the next memory is fully determined. This is captured by the deterministic next-memory function: $\mu(hao) = N(\mu(h), a, o)$. An alternative view of memories is as a directed graph (see Figure 1), with nodes corresponding to memories, and edges defining the transitions between memories, labeled with action/observation pairs. We will discuss the relation between memories and the state space in detail in the next section.

The update parameters $M^{\mu(h)}_{ao}$ transform the current prediction vector that contains predictions for μ-core tests $Q^{\mu(h)}$ in history h to the prediction vector for μ-core tests $Q^{\mu(hao)}$ in history hao. The prediction vector update for mPSRs, upon taking action a in history h and observing o is

$$p(Q^{\mu_j}|hao) = \frac{p(aoQ^{\mu_j}|h)}{p(ao|h)} = \frac{p(Q^{\mu_i}|h)^T M^{\mu_i}_{ao}}{p(Q^{\mu_i}|h)^T m^{\mu_i}_{ao}} \quad (7)$$

where $\mu(h) = \mu_i$ and $\mu(hao) = \mu_j$. The matrix $M^{\mu_i}_{ao}$ is of size $(|Q^{\mu_i}| \times |Q^{\mu_j}|)$ and the vector $m^{\mu_i}_{ao}$ is of size $(|Q^{\mu_i}| \times 1)$. We note here that a vector equivalent to the PSR immediate reward vector (Equation 6) is defined for mPSRs, and is dependent on both the memory μ and action a as follows

$$r^{\mu,a} = \sum_r \sum_{o \in \mathcal{O}^r} r \cdot m^{\mu}_{ao}. \quad (8)$$

Table 1: POMDP, PSR, and mPSR statistics for dynamical systems. Shown are the number of POMDP nominal-states, PSR core tests, mPSR μ-core tests for each memory (the number of memories is the number of values), and the size of the PSR and mPSR models in terms of the number of parameters required to specify the model.

System	POMDP nominal-states	PSR tests	mPSR tests	PSR size	mPSR size
Cheese	11	11	1,1,1,1,2,2,3	3696	792
Network	7	7	4, 6	2912	3160
Shuttle	8	7	1,1,2,2,4	1344	780
4x3	11	11	1,1,1,1,3,4	7392	1892
4x4	16	16	1, 15	2176	1152
Tiger-grid	33	33	1, 32	185130	174570

There is one special form that a memory may take on, and that is when the memory itself serves as state, or equivalently (James, Singh, & Wolfe, 2005), when the memory has only one corresponding μ-core test. These memories are called landmark memories, or simply *landmarks*. The presence of landmarks in a model of a dynamical system can be used advantageously by planning algorithms, as will be discussed in the section on memories and planning.

For this research, the mPSRs used a refined version of the most-recent observation as memories. The refinement was to combine most-recent observations that satisfy both

- have the same sets of μ-core tests, and
- have the same transitions to next memories, for all a, o.

For example, the graph in Figure 1 has two memories, μ_1 corresponds to the situations where the most recent observation is o_1 or o_3, and μ_2 corresponds to the situation where the most recent observation is o_2.

Memories and State Space

In PSRs, the set of all possible prediction vectors defines the state space for the dynamical system (as does the set of all possible POMDP belief-states), but for mPSRs, the state space is defined by both the possible memories and prediction vectors for each of those memories. The μ-core tests for a given memory are a subset of the PSR core tests, and so the possible states corresponding to that memory are subspaces of the entire state space.

Memories do not necessarily partition the state space into disjoint spaces, because different memories may contain the same μ-core test, or even all of the same μ-core tests. However, as noted above, it is generally better to choose memories to overlap as little as possible, and the mPSR algorithm above attempts to do this. This fact will be leveraged during the planning algorithm, where this division of the state space will give computational and space advantages.

Planning with PSRs

The adaptation of point-based methods to PSRs is straightforward. The value function V_n has been shown (James, Singh, & Littman, 2004) to remain a piecewise linear function, but over prediction vectors instead of belief-states. The set P of points is composed of prediction vectors, and so the upper surface of V_n at prediction vector $p(Q|h)$ for history h is given by

$$V_n(p(Q|h)) = \max_{\alpha \in S_n} p(Q|h)^T \alpha \tag{9}$$

which uses Equations 6, 3, 4, and 9 to calculate the value

$$\begin{aligned} V_n(p(Q|h)) &= \qquad\qquad\qquad\qquad\qquad\qquad (10) \\ &= \max_a \Big(R(h,a) + \\ &\qquad \gamma \sum_{o \in \mathcal{O}} pr(o|a,h) \max_{\alpha \in S_n} p(Q|hao)^T \alpha \Big) \\ &= \max_a \Big(p(Q|h)^T r^a + \\ &\qquad \gamma \sum_{o \in \mathcal{O}} p(Q|h)^T m_{ao} \max_{\alpha \in S_n} \frac{p(Q|h)^T M_{ao}}{p(Q|h)^T m_{ao}} \alpha \Big) \\ &= \max_a \Big(p(Q|h)^T r^a + \\ &\qquad \gamma \sum_{o \in \mathcal{O}} \max_{\alpha \in S_n} p(Q|h)^T M_{ao} \alpha \Big) \end{aligned}$$

and so the backup operator returns the vector α that maximizes the reward for step $n+1$ for prediction vector $p(Q|h)$, and is given by

$$\begin{aligned} \mathit{backup}(p(Q|h)) = \operatorname*{argmax}_a \Big(& p(Q|h)^T r^a + \\ & \gamma \sum_{o \in \mathcal{O}} \operatorname*{argmax}_{\alpha \in S_n} p(Q|h)^T M_{ao} \alpha \Big) \end{aligned} \tag{11}$$

So $\alpha' = r^a + \gamma \sum_{o \in \mathcal{O}} M_{ao} \alpha$ for appropriate r^a and α. Given this backup operator, and using prediction vectors for points, the PERSEUS algorithm may then be applied, as well as other point-based value iteration algorithms.

Planning with mPSRs

To take advantage of memories in point-based planning, PSR-PERSEUS was modified for mPSRs to keep a separate set S_n^μ of α vectors for each memory μ, as well as separate set P^μ of points. S_n^μ defines the value function for memory μ's subspace at step n. This (possibly overlapping) partitioning of both the state space and of the value function have potential advantages for planning, and has been shown

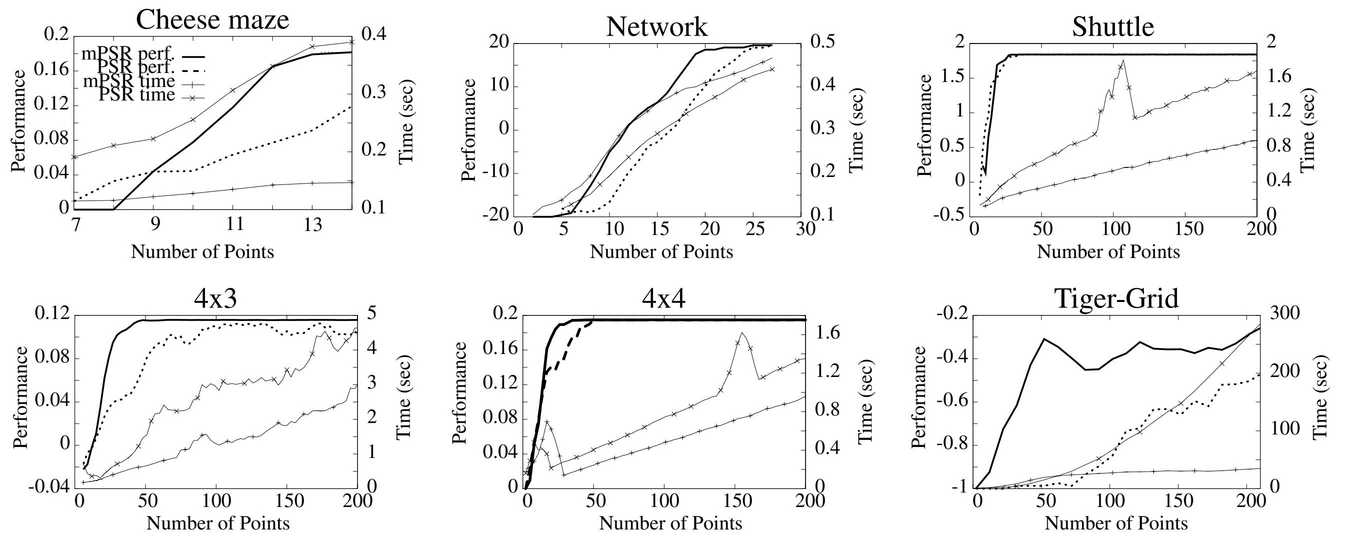

Figure 2: Plots of average reward and time used (y-axis and y2-axis) vs. the number of points used for planning (x-axis) for the mPSR-PERSEUS and PSR-PERSEUS planning algorithms. See text for experimental setup and details.

(James & Singh, 2005) to retain an exact representation of the value function (for the exact value-iteration case).

Each of the memories corresponds to a lower-dimensional state subspace, which can more easily be covered with a given number of points. Further, because each of these subspaces is of lower dimension, the vectors in S_n^μ are of smaller size, and can be stored and computed more quickly and efficiently. Finally, because the value function for a given memory is specific only to the relevant portion of the state space, it will require fewer vectors to represent (as the corresponding policy is expected to be less complex). This reduces the number of vectors needed for comparison when choosing a maximal vector, and so reduces the amount of computation required.

However, it is not guaranteed that these factors will always result in better performing algorithms. There can be overlap in the portions of the state space that is covered, resulting in redundant computation of the value function. This is often due to poor selection of memories, and it is possible that a given dynamical system will not have a memory structure that results in good splitting of the state space.

Memories and Planning

Given separate sets S_n^μ and P^μ for each memory, the backup operator will be modified, which gives insights into some of the advantages of using memories. The backup is defined for memory/prediction vector pairs, and so, for each possible outcome (observation) following the initial action, the next memory is defined by $N(\mu, a, o) = \mu'$. Because the maximization is with respect to memory μ', only the vectors in the set $S_n^{\mu'}$ need be considered. This may be result in a significant speedup, because $S_n^{\mu'}$ may be much smaller than S_n for the non-memory case. The mPSR backup for a given memory μ and prediction vector $p(Q^\mu|h)$ corresponding to history h is

$$backup(\mu, p(Q^\mu|h)) = \underset{a}{\operatorname{argmax}} \Big(p(Q^\mu|h)^T r^{\mu,a} + \gamma \sum_{o\in\mathcal{O}} \underset{\alpha\in S_n^{N(\mu,a,o)}}{\operatorname{argmax}} p(Q^\mu|h)^T M_{ao}^\mu \alpha \Big) \quad (12)$$

where the resulting vector $\alpha_{n+1}^\mu = backup(\mu, p(Q^\mu|h))$ approximately maximizes the reward at time $n+1$ for memory μ and prediction vector $p(Q^\mu|h)$. The only other adaptation to the planning algorithm is that updates must be performed on S_n^μ for all memories in each iteration (in parallel or in a Gauss-Seidel fashion).

For some dynamical systems, there exist landmark memories which require only a single core test, the prediction of which always (James & Singh, 2005) takes on the same value whenever that memory is encountered, and so requires only a single vector α to represent its value function. This is advantageous in planning because a single point can be assigned to this memory, and so there is no need to maximize over α when this memory is encountered.

To give some intuition of this, a dynamical system with all landmark memories is just an MDP, and so only a single point (value) is required for each memory (the memory is state for landmarks). In this case, many planning algorithms when applied to such mPSRs will simply become value iteration on the MDP.

Results

To evaluate the performance difference made by using memories in planning, we conducted experiments on six commonly used dynamical systems, statistics about which are found in Table 1. System definitions in POMDP format can be found at (Cassandra, 1999). PSR models of these systems were constructed according to the algorithm in Littman, Sutton, & Singh (2001), and mPSR models of this system were constructed using the most recent observation as memory and then combining memories as discussed at the end of the mPSR section of this paper.

For each dynamical system, POMDP-PERSEUS, PSR-PERSEUS, and mPSR-PERSEUS were run with different quantities of points specified, starting with the number of memories. For mPSR-PERSEUS, the points were distributed among the various memories in proportion to the dimension of the associated state space (i.e. the number of associated core tests), while using the fact that landmark memories only need a single point. To select points for each memory, the PBVI approach (Pineau, Gordon, & Thrun, 2003) was used: simulating the dynamical system and choosing encountered points that are sufficiently far away from already selected points. For these experiments, the points were required to be only 10^{-4} from all other points, for at least one dimension.

For each quantity of points, the algorithms were run 20 times, and for each run, the time spent on the planning algorithm and the performance of the resulting policy were recorded and averaged. The planning algorithm was run for 500 steps in all cases, and the performance was calculated as the average reward per time step over $100,000$ steps following the resulting optimal policy, while starting from the same state as in training.

The graphs in Figure 2 each show both the average reward (thick lines, solid for mPSR and dashed for PSRs), as well as the time used (thin lines with $+$ for mPSRs, and $\times$ for PSRs). The average reward performance of POMDP-PERSEUS was very similar to that of PSR-PERSEUS, with normalized (by the range of average rewards) RMSD greater than 0.1 on only two systems: Cheese-Maze and Tiger-Grid. Further, for those two systems, the mPSR-PERSEUS performance was significantly (notice the small scale of average rewards for these systems) better than POMDP-PERSEUS by an average (across all quantities of points) of 0.032 on Cheese-Maze and by an average of 0.27 on Tiger-Grid. The time used by POMDP-PERSEUS was also comparable or slower than PSR-PERSEUS. For clarity of presentation, we omit POMDP-PERSEUS from the graphs.

In four of the six systems, mPSR-PERSEUS significantly outperformed PSR-PERSEUS and POMDP-PERSEUS in terms of average reward, and for the other two systems mPSR-PERSEUS performed approximately the same or only slightly better[2]. In terms of timing, on five of the six systems, mPSR-PERSEUS outperformed PSR-PERSEUS and POMDP-PERSEUS, but was outperformed on the sixth (Network). Most significantly, on the largest system (Tiger-Grid), mPSR-PERSEUS significantly outperformed PSR-PERSEUS and POMDP-PERSEUS in terms of both average reward for all numbers of points, and time used to plan (notice the upward trending time line for PSR-PERSEUS, while the time used for mPSR-PERSEUS remains relatively flat).

[2]In the results shown for Tiger-Grid, not enough points were used to have either algorithm perform to near-optimal, but the results do demonstrate how mPSR-PERSEUS has significant advantages with relatively few points

Concluding remarks

The use of memories in planning had previously been shown to have benefits for exact value-iteration, and this paper demonstrates that similar benefits exist for approximate, point-based methods. The benefits have been demonstrated empirically, and the reasons behind them have been explored theoretically. Of course, the amount of benefit depends on the relation between memories and the state space. Selection of proper memories is therefore crucial, and an interesting research topic. Further work should investigate how memories can be used in the selection of the initial-state for point-based value-iteration, similar to the investigation of how PSRs can be used in Izadi, Rajwade, & Precup (2005).

References

Cassandra, A. 1999. Tony's pomdp page. http://www.cs.brown.edu/research/ai/ pomdp/index.html.

Izadi, M. T.; Rajwade, A. V.; and Precup, D. 2005. Using core beliefs for point-based value iteration. In *IJCAI*.

Jaeger, H. 2000. Observable operator processes and conditioned continuation representations. *Neural Computation* 12(6):1371–1398.

James, M. R., and Singh, S. 2005. Planning in models that combine memory with predictive representations of state. In *20th National Conference on Artificial Intelligence*.

James, M. R.; Singh, S.; and Littman, M. L. 2004. Planning with predictive state representations. In *The 2004 International Conference on Machine Learning and Applications*.

James, M. R.; Singh, S.; and Wolfe, B. 2005. Combining memory and landmarks with predictive state representations. In *The 2005 International Joint Conference on Artificial Intelligence*.

Littman, M. L.; Sutton, R. S.; and Singh, S. 2001. Predictive representations of state. In *Advances In Neural Information Processing Systems 14*.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. 18th Int. Joint Conf. on Artificial Intelligence*.

Shani, G.; Brafman, R.; and Shimony, S. 2005. Model-based online learning of POMDPs. In *ECML*.

Singh, S.; James, M. R.; and Rudary, M. R. 2004. Predictive state representations, a new theory for modeling dynamical systems. In *20th Conference on Uncertainty in Artificial Intelligence*.

Spaan, M., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research* 24:195–220.

Sutton, R. S., and Tanner, B. 2004. Temporal-difference networks. In *Advances in Neural Information Processing Systems 17*.

Wolfe, B.; James, M. R.; and Singh, S. 2005. Learning predictive state representations in dynamical systems without reset. In *22nd International Conference on Machine Learning*.