TITLE GOES HERE

Anonymous Submission

Abstract

Recent years have seen the development of efficient and provably correct spectral algorithms for learning models of partially observable environments arising in many applications. But despite the high hopes raised by this new class of algorithms, their practical impact is still below expectations. One reason for this is the difficulty in adapting spectral methods to exploit structural constraints about different target environments which can be known beforehand. A natural structure intrinsic to many dynamical systems is a multi-resolution behavior where interesting phenomena occur at different time scales during the evolution of the system. In this paper we introduce the multi-step predictive state representation (M-PSR) and an associated learning algorithm that finds and leverages frequent patterns of observations at multiple scales in dynamical systems with discrete observations. We perform experiments on robot exploration tasks in a wide variety of environments and conclude that the use of M-PSR improves over the classical PSR for varying amounts of data, environment sizes, and number of observations symbols.

Introduction

Learning models of partially observable dynamical systems is very important in practice — several alternatives...

General algorithms are not designed to exploit frequen patterns/structure in sequences of observations to speed-up learning — but in practice with large observations and highly structured environments this might be necessary to achieve decent results

We propose a new model of predictive state representation for environments with discrete observations: the multi-step PSR (M-PSR)

We show how the standard spectral learning for PSR extends to M-PSR

Then we present a data-driven algorithm for selecting a particular M-PSR from data sampled from a structured partially observable environment

We evaluate the performance of our algorithms in an extensive collection of synthetic environments and conclude that...

The Muti-Step PSR

A linear *predictive state representation* (PSR) for an autonomous dynamical system with discrete observations is a tuple $\mathcal{A} = \langle \Sigma, \boldsymbol{\alpha}_\lambda, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma \in \Sigma} \rangle$ where: Σ is a finite set of possible observations, $\boldsymbol{\alpha}_\lambda, \boldsymbol{\alpha}_\infty \in \mathbb{R}^n$ are vectors of initial and final weights, and $\mathbf{A}_\sigma \in \mathbb{R}^{n \times n}$ are the transition operators associated with each possible observation. The dimension n is the number of states of $\mathcal{A}$. Formally, a PSR is a *weighted finite automata* (WFA) (**?**) computing a function given by the probability distribution of sequences of observations in a partially observable dynamical system with finite state. The function $f_\mathcal{A} : \Sigma^\star \to \mathbb{R}$ computed by $\mathcal{A}$ is given by

$$f_\mathcal{A}(x) = f_\mathcal{A}(x_1 \cdots x_t) = \boldsymbol{\alpha}_\lambda^\top \mathbf{A}_{x_1} \cdots \mathbf{A}_{x_t} \boldsymbol{\alpha}_\infty = \boldsymbol{\alpha}_\lambda^\top \mathbf{A}_x \boldsymbol{\alpha}_\infty \quad .$$

The value of $f_\mathcal{A}(x)$ is interpreted as the probability that the system produces the sequence of observations $x = x_1 \cdots x_t$ starting from the initial state specified by $\boldsymbol{\alpha}_\lambda$.

To define our model for multi-step PSR we basically augment a PSR with two extra objects: a set of *multi-step observations* $\Sigma' \subset \Sigma^+$ containing non-empty strings formed by basic observations, and a *coding function* $\kappa : \Sigma^\star \to \Sigma'^\star$ that given a string of basic observations produces an equivalent string composed using multi-step observations. The choice of Σ' and κ can be quite application-dependent, in order to reflect the particular patterns arising from different environments. However, we assume this objects satisfy a basic set of requirements for the sake of simplicity and to avoid degenerate situations:

1. The set Σ' must contain all symbols in Σ; i.e. $\Sigma \subseteq \Sigma'$
2. The function κ satisfies $\partial(\kappa(x)) = x$ for all $x \in \Sigma^\star$, where $\partial : \Sigma'^\star \to \Sigma^\star$ is the *decoding morphism* between free monoids given by $\partial(z) = z \in \Sigma^\star$ for all $z \in \Sigma'$. Note this implies that $\kappa(\epsilon) = \epsilon$, $\kappa(\sigma) = \sigma$ for all $\sigma \in \Sigma$, and κ is injective.

Using these definitions, a *multi-step PSR* (M-PSR) is a tuple $\mathcal{A}' = \langle \Sigma, \Sigma', \kappa, \boldsymbol{\alpha}_\lambda, \boldsymbol{\alpha}_\infty, \{\mathbf{A}_\sigma\}_{\sigma \in \Sigma'} \rangle$ containing a PSR with observations in Σ', together with the basic observations Σ and the corresponding coding function κ.

Borja: For now, this is enough

Examples

We now describe several examples of M-PSR, and put special emphasis on models that will be used in our experiments.

A PSR with a single observation $\Sigma = \{a\}$ can be used to measure the time – i.e. number of discrete time-steps – until a certain event happens (**?**). In this case, a natural approach to build an M-PSR for timing models is to build a set of multi-step observations containing sequences whose lengths are powers of a fixed base. That is, given an integer $b > 0$, we build the set of multi-step observations as $\Sigma' = \{a, a^b, a^{b^2}, \ldots, a^{b^K}\}$ for some positive K. A natural choice of coding map in this case is the one that represents any length t as a number in base b, with the difference that the largest power b that is allowed is b^K. This corresponds to writing (in a unique way) $t = t_0 b^0 + t_1 b^1 + t_2 b^2 + \cdots + t_K b^K$, where $0 \leq t_k \leq b-1$ for $0 \leq k \leq K-1$, and $t_K \geq 0$. With this decomposition we obtain the coding map $\kappa(a^t) = (a^{b^K})^{t_K} (a^{b^{K-1}})^{t_{K-1}} \cdots (a^b)^{t_1} (a)^{t_0}$. Note that we choose to write powers of longer multi-step observations first, followed by powers of shorter multi-step observations. For further reference, we will call this model the *base M-PSR* (with base b and largest power K) for modelling distributions over time.

Borja: Re-wrote the presentation of the base M-PSR a little bit

For the multiple observation case one can also use a Base M-PSR. In this case $\Sigma' = \{\sigma_1, \sigma_2, \sigma_1^b, \sigma_2^b, \sigma_1^{b^2}, \sigma_2^{b^2}, \ldots, \sigma_1^{b^k}, \sigma_2^{b^k}\}$. Here σ_1 and σ_2 are two observations symbols' this can of course be extended for any finite number of observations. For the encoding map κ we first split the string into sequences of a fixed symbol and then use the same encoding as for timing. As an example: $\kappa(\sigma_1^5 \sigma_2^3) = (\sigma_1^4)(\sigma_1)(\sigma_2^2)(\sigma_2)$.

Another example for the multiple observation case is what we call a **Tree M-PSR**. For the Tree M-PSR, we set $\Sigma' = \{s \in \Sigma^\star, len(s) <= L\}$. Here L is a parameter of choice, but note that $|\Sigma'| = |\Sigma|^L$, thus in practice L must remain small as learning operators is computationally intensive. For the decoding map κ, we first split a string x as $x = x_1 x_2 \ldots x_n x_f$, where $len(x_i) == L, \forall i <= n$ and $len(x_f) = len(x) - (n \cdot L)$. With this we set $\kappa(x) = \{x_1, x_2, \ldots, x_n, x_f\}$.

The constructions of the M-PSRs above are not dependent on the environment. In our results section, we find that performance of M-PSRs depend heavily on how Σ' reflects the observations in one's environment. Thus, we develop an algorithm for choosing Σ'. In addition, we provide a κ which one can apply to any M-PSR and which delivers good experimental performance. Together, these yield another type of M-PSR, which we call a **Data-Driven M-PSR**.

Lucas: Made modifications to examples to replicate changes made by borja

Learning Algorithm for M-PSR

In this section, we describe a learning algorithm for M-PSR which combines the standard spectral algorithm for PSR (**?**) with a data-driven greedy algorithm for building an extended set of symbols Σ' containing frequent patterns that minimise a coding cost for a general choice of coding function κ.

Spectral Learning Algorithm

We extend (**?**) to M-PSR under the assumption that κ and Σ' are given.

Borja: Need to fill this. Probably copy-paste from the ODM paper will do.

Borja: BEGIN OF COPY-PASTE. NEEDS EDITS

A convenient algebraic way to summarize all the information conveyed by f is with its *Hankel matrix*, a bi-infinite matrix $\mathbf{H}_f \in \mathbb{R}^{\Sigma^\star \times \Sigma^\star}$ with rows and columns indexed by strings in $\Sigma^\star$. Strings indexing rows and columns are interpreted as prefixes and suffixes respectively. The entries in $\mathbf{H}_f$ are given by $\mathbf{H}_f(u,v) = f(u,v)$ for every $u, v \in \Sigma^\star$.

Although $\mathbf{H}_f$ is an infinite matrix, in some cases it can have finite rank. In particular, a well-known result states that $\mathbf{H}_f$ has rank at most n if and only if there exists a PSR $\mathcal{A}$ with n states satisfying $f_\mathcal{A} = f$ (**?**; **?**). This result is the basis of recently developed spectral learning algorithms for PSRs (**?**), which we review in Sec. **??**.

Instead of looking at the full Hankel matrix, algorithms usually work with finite sub-blocks of this matrix. A convenient way to specify such blocks is to give the "names" to the rows and columns. Specifically, given a finite set of prefixes $\mathcal{P} \subset \Sigma^\star$ and a finite set of suffixes $\mathcal{S} \subset \Sigma^\star$, the pair $\mathcal{B} = (\mathcal{P}, \mathcal{S})$ is a *basis* defining the sub-block $\mathbf{H}_\mathcal{B} \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ of $\mathbf{H}_f$, whose entries are given by $\mathbf{H}_\mathcal{B}(u,v) = \mathbf{H}_f(u,v)$. Note that every sub-block built in this way satisfies $\mathrm{rank}(\mathbf{H}_\mathcal{B}) \leq \mathrm{rank}(\mathbf{H}_f)$; when equality is attained, the basis $\mathcal{B}$ is *complete*.

Sometimes it is also convenient to look at one-step shifts of the finite Hankel matrices. Let $\mathbf{H} \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ be a finite sub-block of $\mathbf{H}_f$ specified by a basis $\mathcal{B} = (\mathcal{P}, \mathcal{S})$. Then, for every symbol $\sigma \in \Sigma$, we define the sub-block $\mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P} \times \mathcal{S}}$ whose entries are given by $\mathbf{H}_\sigma(u,v) = \mathbf{H}_f(u, \sigma v)$. For a fixed basis, we also consider the vectors $\mathbf{h}_\mathcal{S} \in \mathbb{R}^\mathcal{S}$ with entries given by $\mathbf{h}_\mathcal{S}(v) = \mathbf{H}_f(\lambda, v)$ for every $v \in \mathcal{S}$, and $\mathbf{h}_\mathcal{P} \in \mathbb{R}^\mathcal{P}$ with $\mathbf{h}_\mathcal{P}(u) = \mathbf{H}_f(u, \lambda)$.

The Hankel matrix $\mathbf{H}_f$ is tightly related to the *system dynamics matrix* (SDM) of the stochastic process described by f (**?**), but while the entries of the Hankel matrix represent *joint* probabilities over prefixes and suffixes, the corresponding entry in the SDM is the *conditional* probability of observing a suffix given the prefix.

The algorithm takes as input Σ and a basis $\mathcal{B}$ in $\Sigma^\star$, uses them to estimate the corresponding Hankel matrices, and then recovers a PSR by performing singular value decomposition and linear algebra operations on these matrices. Although the method works almost out-of-the-box, in practice the results tend to be sensitive to the choice of basis. Thus, after briefly recapitulating how the spectral learning algorithm proceeds, we will devote the rest of the section to describe a procedure for building a basis which is tailored for the case of learning option duration models.

Suppose the basis $\mathcal{B}$ is fixed and the desired number of states n is given. Suppose that a set of sampled trajectories was used to estimate the Hankel matrices $\mathbf{H}, \mathbf{H}_\sigma \in \mathbb{R}^{\mathcal{P}\times\mathcal{S}}$ and vectors $\mathbf{h}_\mathcal{P} \in \mathbb{R}^\mathcal{P}$, $\mathbf{h}_\mathcal{S} \in \mathbb{R}^\mathcal{S}$ defined in Sec. **??**. The algorithm starts by taking the truncated SVD $\mathbf{U}_n\mathbf{D}_n\mathbf{V}_n^\top$ of $\mathbf{H}$, where $\mathbf{D}_n \in \mathbb{R}^{n\times n}$ contains the first n singular values of $\mathbf{H}$, and $\mathbf{U}_n \in \mathbb{R}^{\mathcal{P}\times n}$ and $\mathbf{V}_n \in \mathbb{R}^{\mathcal{S}\times n}$ contain the first left and right singular vectors respectively. Finally, we compute the transition operators of a PSR as $\mathbf{A}_\sigma = \mathbf{D}_n^{-1}\mathbf{U}_n^\top\mathbf{H}_\sigma\mathbf{V}_n$, and the initial and final weights as $\boldsymbol{\alpha}_\lambda^\top = \mathbf{h}_\mathcal{S}^\top\mathbf{V}_n$ and $\boldsymbol{\alpha}_\infty = \mathbf{D}_n^{-1}\mathbf{U}_n^\top\mathbf{h}_\mathcal{P}$. This yields a PSR with n states. It was proved in (**?**) this algorithm is statistically consistent: if the population Hankel matrices are known and the basis $\mathcal{B}$ is complete, then the learned PSR is equivalent to the one that generated the data.

Borja: END OF COPY-PASTE

Notation

Borja: We should move this into the two subsections below

Obs: A mapping from observation sequences to the number of occurrences of that sequence in one's dataset.

SubObs : all substrings of **Obs**.

Q: A query string (a string for which one wishes to determine the probability of).

bestE: A map from indices i of Q to the optimal encoding of Q[:i].

minE: A map from indices i of Q to $|bestE[i]|$

opEnd: A map from indices i of Q to the set of strings in Σ': $\{x \in \Sigma' s.t Q[i - x.length : i] == x\}$

numOps: The desired number of operators one wants in Σ'. I.e numOps = $|\Sigma'|$

A General Coding Function

Here we provide a dynamic programming algorithm which can serve as κ for any M-PSR. Given a query string Q, and a set of transition sequences Σ', the algorithm minimizes the number of sequences used in the partition $\kappa(Q)$. In other words, the algorithm minimizes $|\kappa(Q)|$. For the single observation case, the algorithm is equivalent to the coin change problem.

For a given string Q, the algorithm inductively computes the optimal string encoding for the prefix Q[:i]. It does so by minimizing over all $s \in \Sigma'$ which terminate at the index i of Q.

Greedy Selection of Multi-Step Observations

Here we present a greedy heuristic which learns the multi-step transition sequences Σ' from observation data. Having a Σ' which reflects the types of observations produces by one's system will allow of short encodings when coupled with our a dynamic programming encoding algorithm. In practice, this greedy algorithm will pick substrings from one's observation set which are long, frequent, and diverse. From an intuitive standpoint, one can view structure in observation sequences as relating to the level of entropy in the system's observations.

Algorithm 1 Encoding Algorithm

```
1:  procedure DPENCODE
2:      $bestE[] \leftarrow newString[len(Q)+1]$
3:      $minE[] \leftarrow newInt[len(Q)+1]$
4:      $opEnd[] \leftarrow newString[len(Q)+1][]$
5:      $bestEnd[0] = Q[0]$
6:      $minE[0] = 0$
7:      for i in range[1,Q.length] do
8:          $opEnd[i] \leftarrow \{s \in \Sigma', Q[i - len(s) : i] == s\}$
9:      end for
10:     for i in range[1,Q.length] do
11:         $bestOp \leftarrow null$
12:         $m \leftarrow null$
13:         for $s \in opEnd[i]$ do
14:             $tempInt \leftarrow minE[i - len(s)] + 1$
15:             if $m == null$ or $tempInt < m$ then
16:                 $m \leftarrow temp$
17:                 $bestOp \leftarrow s$
18:             end if
19:         end for
20:         $minE[i+1] \leftarrow m$
21:         $bestE[i+1] \leftarrow bestE[i - len(bestOp)] + bestOp$
22:     end for
          return $bestE[len(Q)]$
23: end procedure
```

As a preprocessing step, we reduce the space of substrings textbfsubObs to the k most frequent substrings in our observation set. Here frequent means the number of observation sequences a given substring $s \in subObs$ occurs in.

Lucas: Added substring preprocessing step

The algorithm evaluates substrings based on how much they reduce the number of transition operators used on one's observation data. The algorithm adds the best operator iteratively with Σ' initialized to Σ. More formally at the i'th iteration of the algorithm the following is computed: $min_{sub \in SubObs} \sum_{obs \in Obs} |\kappa(obs, \Sigma'_i \cup sub)|$. The algorithm terminates after the **numOps** iterations.

Experiments

We assess the performance of PSRs and different kinds of Multi-PSRs on labyrinth environments. We look to see how performance varies as parameters are varied. Parameters include the model size, the number of observations used, and the type of environment. For all the plots, the x-axis is model size of the PSR/M-PSRs and the y-axis is an error measurement of the learned PSR/M-PSRs.

Obtaining Observation Sequences

In all the experiments we consider, an agent is positioned in a starting location and stochastically navigates the environment based on transition probabilities between states. When state-to-state transitions occur an observation symbol is produced. When the agent exits the labyrinth, we say the trajectory is finished, and we record the concatenation of the

Algorithm 2 Base Selection Algorithm

```
1: procedure BASE SELECTION
2:     $\Sigma' \leftarrow \{s, s \in \sum\}$
3:     $Subs \leftarrow \{$k frequent $s \in subObs\}$
4:     $prevBestE \leftarrow null$
5:     for each obs in Obs do
6:         $prevBestEncoding[obs] \leftarrow len(obs)$
7:     end for
8:     $i \leftarrow 0$
9:     while $i < numOperators$ do
10:        $bestOp \leftarrow null$
11:        $bestImp \leftarrow null$
12:        for each s $\in Subs$ do
13:            $c \leftarrow 0$
14:            for each obs in Obs do
15:                $c \leftarrow c + DPEncode(obs) - prevBestE(obs)$
16:            end for
17:            if $c > bestImp$ then
18:                $bestOp \leftarrow observation$
19:                $bestImp \leftarrow c$
20:            end if
21:        end for
22:        $\Sigma' \leftarrow \Sigma' \cup bestOp$
23:        for each obs in Obs do
24:            $prevBestE \leftarrow DPEncode(obs, \Sigma')$
25:        end for
26:        $i \leftarrow i + 1$
27:    end whilereturn $\Sigma'$
28: end procedure
```

symbols produced. We call this concatenation the observation sequence for that trajectory.

Learning Implementation: Timing v.s Multiple Symbols

For the timing case, we construct our empirical hankel matrix by taking P,S = $\{\sigma^i, \forall i <= n\}$. The parameter n depends on the application. For Double Loop environments we set n = 150, while for the pacman labyrinth n = 600. For these choices of n, we verify that as the amount of data gets large the learned PSR with the true model size becomes increasingly close to the true model. For Base M-PSR, we set Σ' to be $\sigma^{2^k}, k <= 256$.

For multiple observations a slightly more complex approach is required to choose P and S. For prefixes P, we select the k most frequent prefixes from our observations set. For suffixes S, we take all suffixes that occur from our set of prefixes. We also require prefix completeness. That is if p' is a prefix of p $\in P$, then p' $\in P$. This heuristic for constructing empirical hankel matrices was given in previous work by [] and it showed that (). For **Base M-PSR**, we set Σ' to be $\{x^{2^k}, \forall x \in \Sigma', k <= 256\}$. For the **Tree M-PSR** we set L to 7.

Measuring Performance

For timing, the goal is to make predictions about how long the agent will survive the environment. One can also ask conditional queries such as how long the agent should expect to survive given that t seconds have elapsed

$$f(\sigma^m|\sigma^n) = \frac{\alpha_\lambda \cdot A(\kappa(\sigma^m)) \cdot \alpha_\infty}{\alpha_\lambda \cdot (I - A_\sigma)^{-1} \cdot \alpha_\infty}$$

The goal for the multiple observation labyrinths is to make predictions about seeing observation sequences. Conditional queries are also possible here

$$f(a^{m_1}b^{m_2}|a^{n_1}b^{n_2}) = \frac{\alpha_\lambda \cdot A(\kappa(a^{m_1}b^{m_2})) \cdot \alpha_\infty}{\alpha_\lambda \cdot A(\kappa(a^{n_1}b^{n_2})) \cdot \alpha_\infty}$$

To measure the performance of a PSR/M-PSR we use the following norm:

$$||f - \hat{f}|| = \sqrt{\sum\nolimits_{x \in observations} (f(x) - f(\hat{x}))^2}$$

We use this norm because of a bound presented by [AUTHORS], which states that (). Here the function f denotes the true probability distribution over observations and the function $\hat{f}$ denotes the function associated with the learned M-PSR/PSR. In our environments, the function f is obtainable directly as we have access to the underlying HMMs.

Since the set of observations Σ^* is infinite, we compute approximations to this error norm, by fixing a set of strings T and summing over T. For the timing case, we take T to be the $\{\sigma^k, \forall k <= n\}$, while for the multiple observation case, we take all possible strings producible from the prefixes and suffixes in our dataset. That is, for the multiple observation case $T = \{ps, \forall p \in P, \forall s \in S\}$.

Double Loop Timing

For timing, we start by considering a double loop environment. The lengths of the loops correspond to the number of states in the loop. A trajectory begins with the agent starting at the intersection of the two loops. At the intersection of the two loops, the agent has a 50 percent chance of entering either loop. At intermediate states in the loops the agent moves to the next state in the loop with probability 1-P and remains in its current state with probability P. Here, P represents the self-transition probability for internal states. Exit states are located halfway between each loop. At an exit state, the agent has a 50 percent probability of exiting the environment.

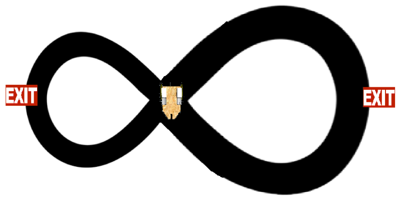

Figure 1: Double Loop Environment

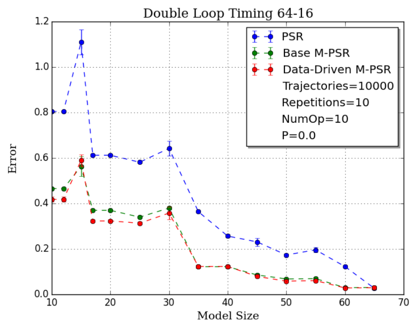

Figure 2: Double Loop 64-16

Number of Trajectories

Here, we vary the number of observations used in our dataset. PSRs/M-PSRs learned in Figure 4 use 100 observation sequences, while those in Figure 5 use 10000. In both cases the M-PSRs outperform the standard PSR for reduced model sizes.

Noise: P

Next, we vary the self-transition probability P to simulate noise in an environment. Figure 5 is a 64-16 double loop with P=0.2, and Figure 6 is a 64-16 double loop with P=0. We find that the noisy loops are more compressible, but the performance is worse for higher models. Nevertheless, M-PSRs still significantly outperform the standard PSR for reduced model sizes.

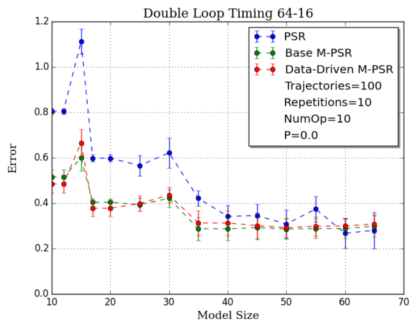

Figure 3: Double Loop 64-16

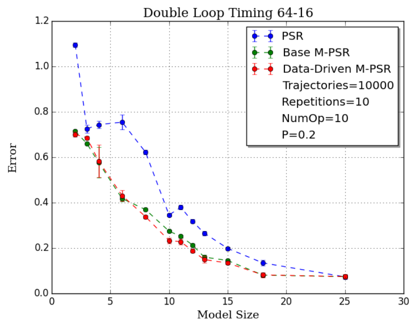

Figure 4: Double Loop 64-16

Loop Lengths

So far we have been using a 64-16 Double Loops. Here observations will come in low multiples of large powers of two. Intuitively, in this case the Base M-PSR should shine the most. In Figure 8, we plot the results of a 47-27 labyrinth where observations will not be so easily expressed from the Base M-PSR. Once again, M-PSRs outperform the standard PSR for reduced model sizes. In addition we see that the Data-Driven M-PSR does better than the Base M-PSR.

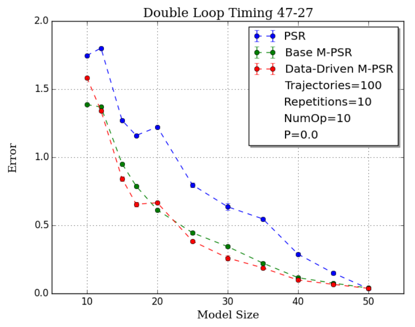

Figure 5: Double Loop 47-27

Choice of K for Base M-PSRs

Below we plot performance of Base M-PSRs with different values of k. We note that as higher powers of operators are included performance improves.

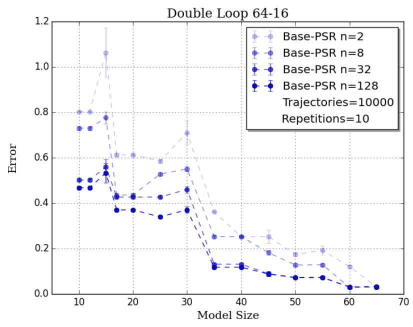

Figure 6: Double Loop 64-16

Varying NumOps

Here, we look at how the varying the number of multi-step transition operators affects performance for the 64-16 double loop. In this environment, a higher number of operators improves performance up until about 20 operators.

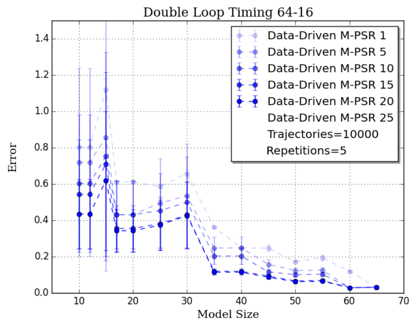

Figure 7: 64-16 Double Loop

Large Labyrinth Timing

We proceed to work with a more complex labyrinth environment. Figure 10 shows it's graphical representation. We test a larger environment as results in such a system would transfer to applications such as pacman. Transitions to new states occur with equal probability. The weight between transitions corresponds to the number of time steps. We add an additional parameter sF: stretchFactor, which multiplies all of the weights in the graph.

Number of Trajectories

In Figures 11 and 12 we vary the number of observations used for learning. First we note that performance of all models is significantly worse for less data. Nevertheless, M-PSRs outperform the traditional PSR regardless.

Stretch Factor: sF

In Figures 13 and 14 we vary the stretch factor parameter. We find that a higher values of sF allow for increased improvement of the M-PSR relative to the performance of the standard PSR.

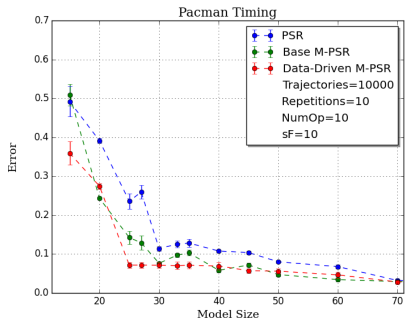

Figure 8: Pacman Labyrinth

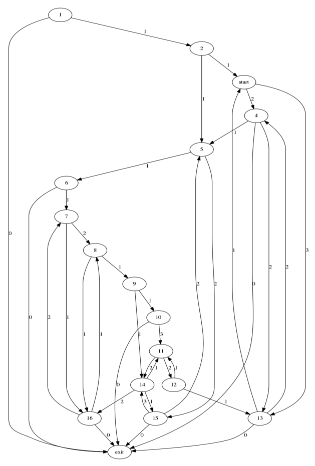

Figure 9: Graph of pacman

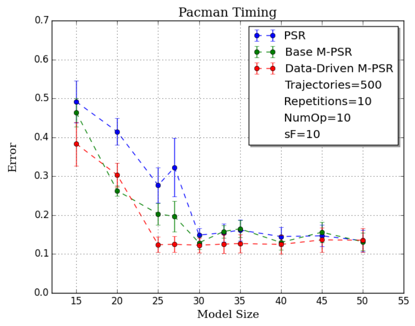

Figure 10: Pacman Labyrinth

Multiple Observations: Colored Loops

We now move to the multiple observation case. Here the Data-Driven M-PSRs really show their strength as observation sequences are more complex. We construct a Double Loop environment where one loop is green and the other is blue. The lengths of each loop are also varied, see Fig-

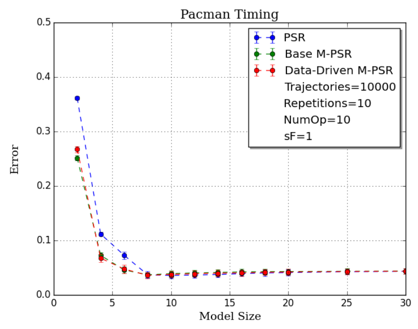

Figure 11: Stretch Factor: 1

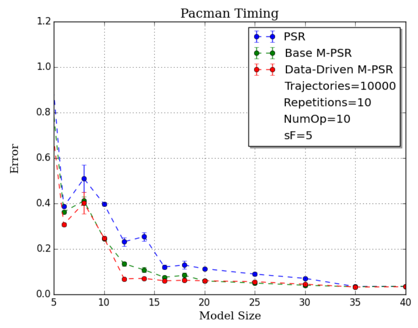

Figure 12: Stretch Factor: 5

ure 4 and Figure 5. We fix the length of observations to be $TrajectoryLength := (len(loop1) + len(loop2)) * 3$. To build empirical estimates of probabilities we set

$$f(x) = \frac{\#occurancesofx}{counts(s \in Obs, len(s) >= x)}$$

This means that the PSRs will compute the probability of x occurring as a prefix.

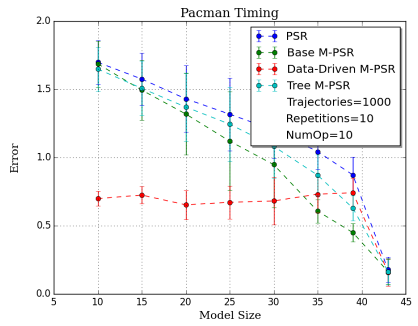

Figure 13: Colored Loops 27-17

Lucas: Removed picture of colored loops. Waste of space and leaves too much evidence of MS paint.

Number of Trajectories

As for the timing case, we vary the amount of data to learn PSRS/M-PSRs in Figures 15 and 16. Once again we find M-PSRs perform far better, especially the Data-Driven M-PSR. This makes sense as when complexity in observations increases only custom M-PSRs will express transitions compactly.

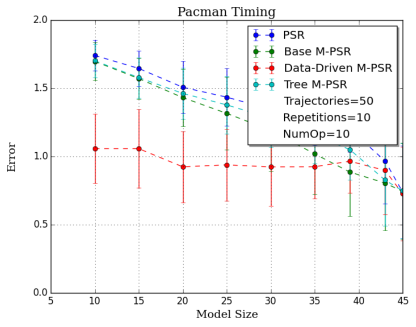

Figure 14: Colored Loops 27-17

Data Driven Sequences

For the double loop case the greedy approach learns multiples of the loop lengths which results in partitions which use fewer operators. As an example, for the 47-27 labyrinth the greedy heuristic picked the following operators: $\Sigma' = \{\sigma^{47}, \sigma^{27}, \sigma^{74}, \sigma^{94}...\}$. For Pacman, the learned strings are multiples of the stretch factor. For Colored Double Loops consistent operators in Σ' are $\{g^{27}, b^{17}, g^{27}b^{17}, b^{17}g^{27}\}$. Figure 19,20,and 21 show performance as we include more operators.

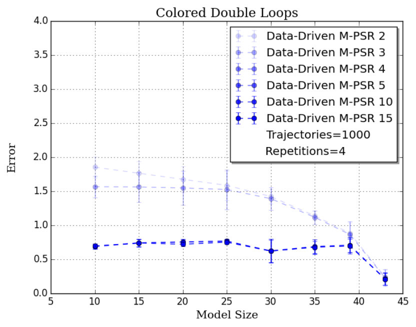

Figure 15: Colored Loops 27-17

Conclusion

Acknowledgments

Funding and friends...

References

Boots, B.; Siddiqi, S.; and Gordon, G. 2011. Closing the learning planning loop with predictive state representations. *International Journal of Robotic Research.*