Learning Multi-Step Predictive State Representations

Lucas Langer, Borja Balle, Doina Precup

July 10, 2016

Predictive State Representations (PSR)

Goals:

1. Make predictions in partially observable environments
2. Learn a representation of hidden states

Motivation:

1. Globally optimal predictions (unlike HMMs)
2. Learn smaller representations

Why Multi-Step PSRs?

1. Leverage structure at different time-scales
2. Learn state transitions for interesting observation sequences

PSR: The single observation case

1. PSR defined by: $\langle \alpha_0, \{A_\sigma\}, \alpha_\infty \rangle$

   where

   α_0 is an initial weighting on states $1xn$

   A_σ is a transition matrix nxn

   α_∞ is a normalizer $nx1$

2. PSRs compute probabilities of observations

   $f(\sigma^k) = \alpha_0 \cdot A_\sigma^k \cdot \alpha_\infty$

Spectral Learning of PSRs

Step 1: Represent Data as a Hankel Matrix

Step 2: Singular Value Decomposition

Step 3: Pick Model Size

Step 4: Linear Algebra

Result: $\langle \alpha_0, \{A_\sigma\}, \alpha_\infty \rangle$

Adding multi-step operators: The Base System

- Idea: Learn $\{A_\sigma, A_{\sigma^2}, A_{\sigma^4}, A_{\sigma^8}, ...A_{\sigma^{2N}}\}$ as extra transition operators

  Note: operators learned separately

  $f(\sigma^{11}) = \alpha_0 \cdot A_{\sigma^8} \cdot A_{\sigma^2} \cdot A_{\sigma^1} \cdot \alpha_\infty$

- **Why might this help?**

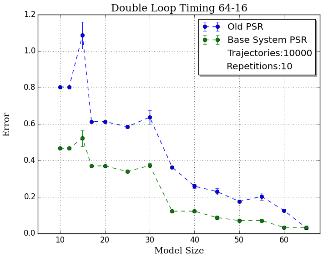
  Reduce error build up

  Capture structure

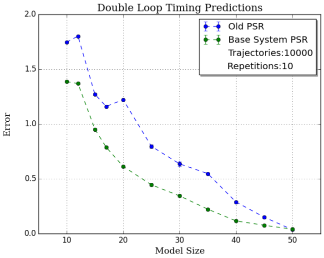
  Faster computations

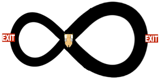
Base System Performance for Loops

64-16 Loop Lengths

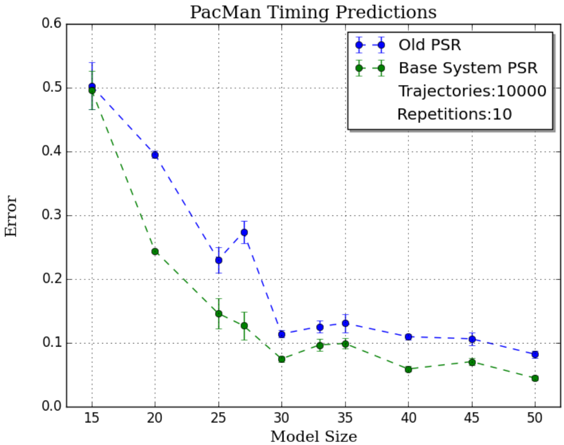


47-27 Loop Lengths

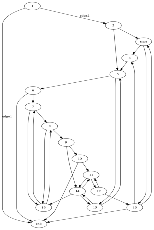




Pacman-like Labyrinth



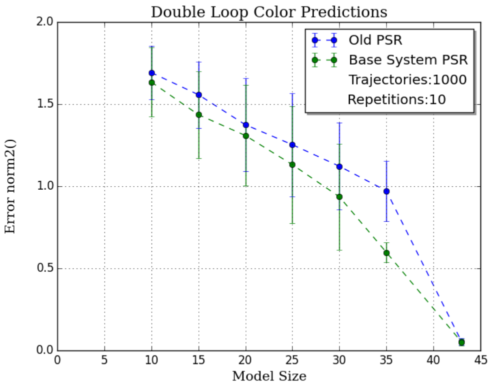
(a) Pacman

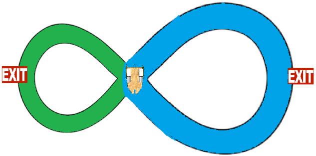


(b) Graph

Wall Color Predictions

We paint the first loop green and the second loop blue





In general, which operators to learn?

- Observations: $\{a^{30}{:}10,\ a^{60}{:}5,\ b^{18}{:}15\}$
  Desired Operators: $A_{a^{30}}$, $A_{b^{18}}$, A_a, A_b
- Substring properties: **long**, **frequent**, **diverse**
  Structured environments should be easier
- Iterative greedy heuristic works well
  Could also try an entropy based approach

How should we execute queries

- **Minimize number of matrices**

  Represent transitions as compactly as possible

- Query string: "abcacb", Operators $= \{A_{ab}, A_{bca}, A_{cb}, A_a, A_b\}$

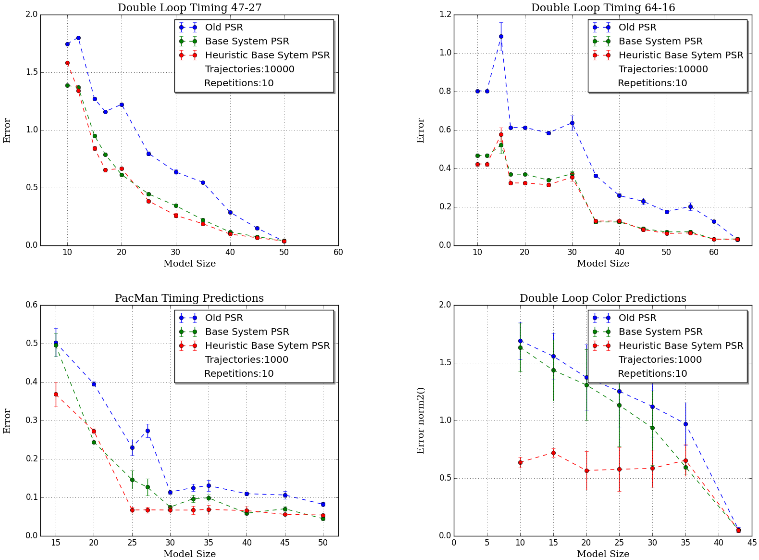
  Desired partition: "a—bca—cb"

  Computation: $f(abcacb) = \alpha_0 \cdot A_a \cdot A_{bca} \cdot A_{cb} \cdot \alpha_\infty$

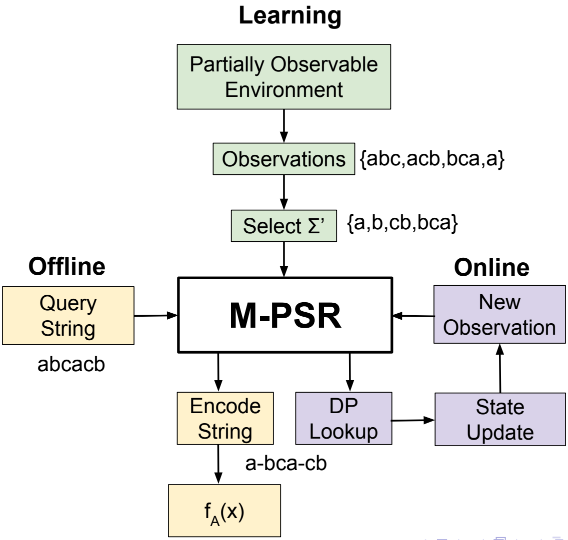
- Solution: dynamic programming

  State update for online applications

Performance of Heuristics



The Big Picture



Questions?