Learning Multi-Step Predictive State Representations

ADD CITATIONS into the presentation Lucas Langer, Borja Balle, Doina Precup

July 11, 2016

Predictive State Representations (PSR)

Goals:

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

Advantages over HMMs

1. Globally optimal predictions
2. Smaller representations of state

Why Multi-Step PSRs?

1. Leverage structure of underlying environment
2. Directly apply transitions which occur at longer time-scales

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 $(nx1)$
   n: number of latent states

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

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

   Operators learned separately so $A_\sigma \cdot A_\sigma \neq A_{\sigma^2}$

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

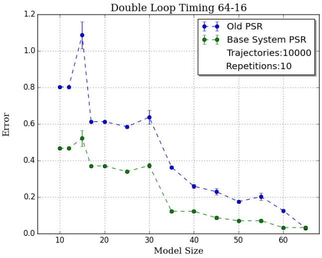
3. **Why might this help?**

   Reduce build up of error

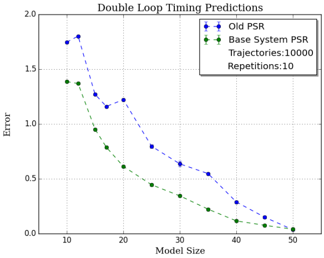
   Faster computations

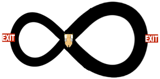
Base System Performance for Loops

64-16 Loop Lengths

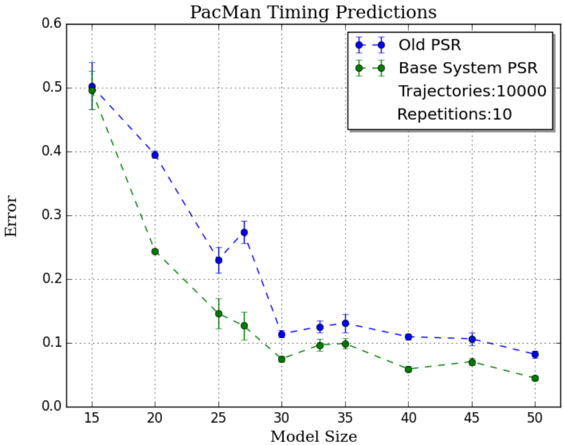


47-27 Loop Lengths

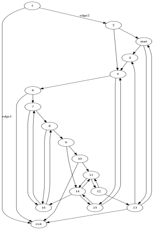




Pacman-like Labyrinth



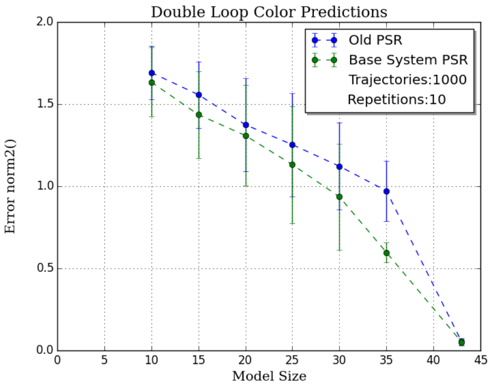
(a) Pacman

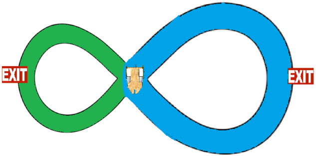


(b) Graph

Wall Color Predictions

We paint the first loop green and the second loop blue





In general, which operators to learn?

1. Observations: $\{a^{30}b^{15},\ a^{60},\ b^{15}\}$

   Desired operators: $A_{a^{30}b^{15}},\ A_{a^{60}},\ A_{a^{30}},\ A_{b^{15}},\ A_a,\ A_b$

2. Sub-string properties: **long**, **frequent**, **diverse**

   Structured environments should be easier

3. Trade-off between number of operators and learning time

   Iterative greedy heuristic works well

   Could also try an entropy based approach

How should we execute queries

1. **Minimize number of matrices**

   Represent transitions as compactly as possible

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

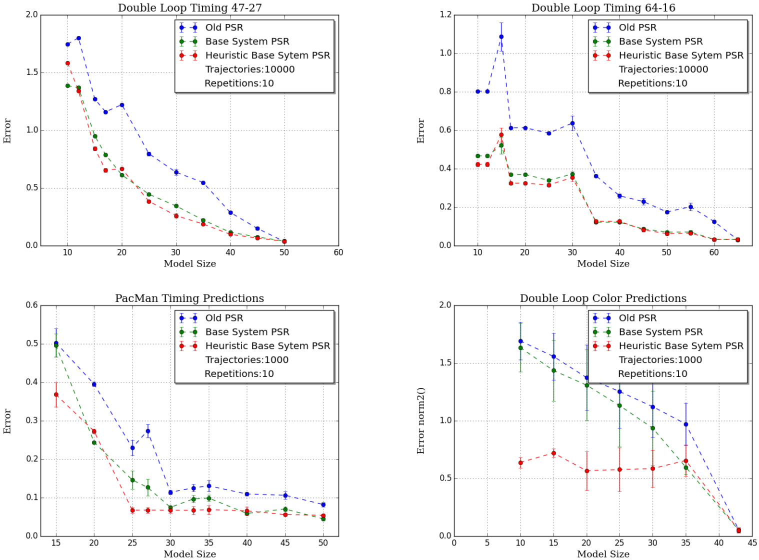
   Desired partition: "a—bca—cb"

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

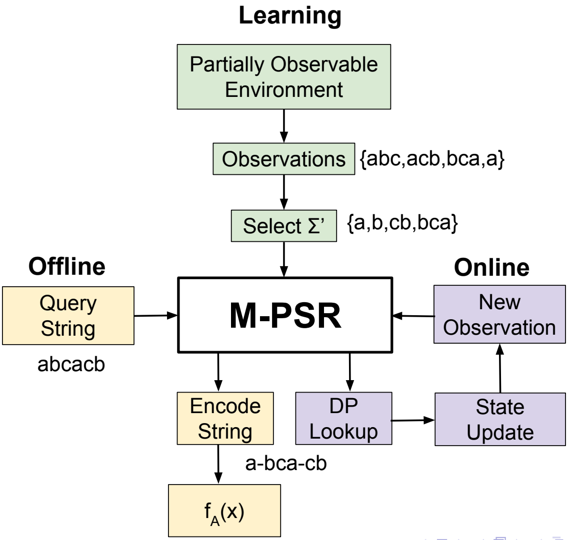
3. Solution: dynamic programming

   State update for online applications

Performance of Heuristics



The Big Picture



Questions?