Spectral learning for structured partially observable environments

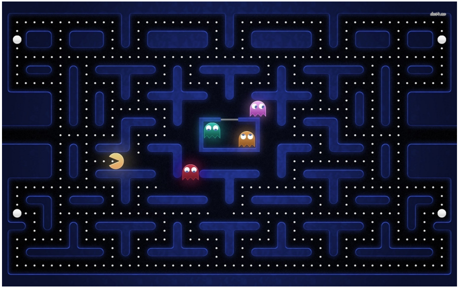
Lucas Langer

Supervisors: Borja Balle, Doina Precup

August 19, 2015

Structured Environments

1. Goal: Predictions
2. Plan: Exploit structure
3. Example: Pacman



PSR: The Timing Case

- Model environment with a Predictive state representation
- For timing we have one observation symbol: σ

  Notation: σ: one time unit, σ^k: k time units

- PSR defined by: $< \alpha_0, \{A_\sigma\}, \alpha_\infty >$

  α_0: Initial weighting on states $1xn$

  A_σ: Transition matrix nxn

  α_∞: Normalizer $nx1$

- PSRs compute probabilities of observations

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

- Example of a PSR: HMM

Spectral Learning of PSRs

Step 1: Represent data as a matrix

Step 2: Singular value decomposition

Step 3: Pick number of states for PSR

Step 4: Learn $< \alpha_0, \{A_\sigma\}, \alpha_\infty >$ with matrix computations

The Base System

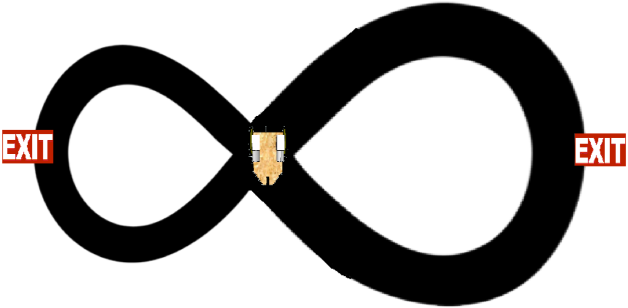
- Idea: Add $\{A_\sigma, A_{\sigma^2}, A_{\sigma^4}, A_{\sigma^8}, ...A_{\sigma^N}\}$ as extra transition operators
- Timing queries: $f(\sigma^{11}) = \alpha_0 * A_{\sigma^8} * A_{\sigma^2} * A_{\sigma^1} * \alpha_\infty$
- Motivation:

  1) Express transitions directly

  2) Faster queries

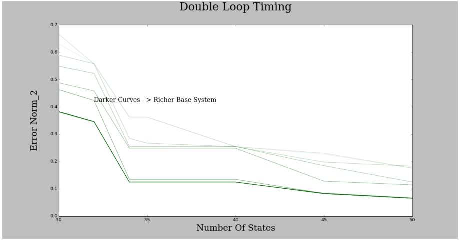
Timing with the Base

Agent goes through loops until leaving through an exit state. Loop lengths are 64 and 16 time units (not to scale).

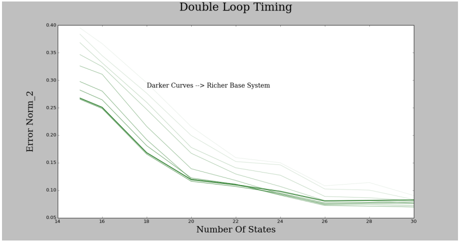


Base System Performance for Loops

No noise in durations



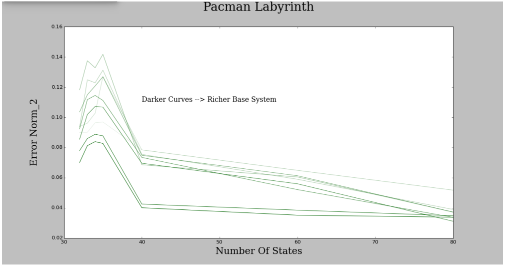
Noise in durations

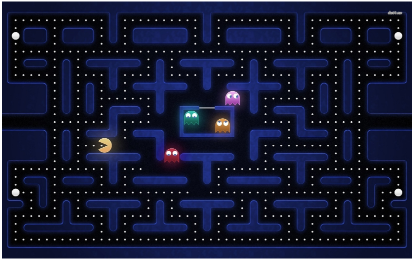


- Noise allows for smaller models
- $||f - \hat{f}|| = \sqrt{\sum_{x \in observations} (f(x) - f(\hat{x}))^2}$

Pacman Labyrinth

Timing Predictions

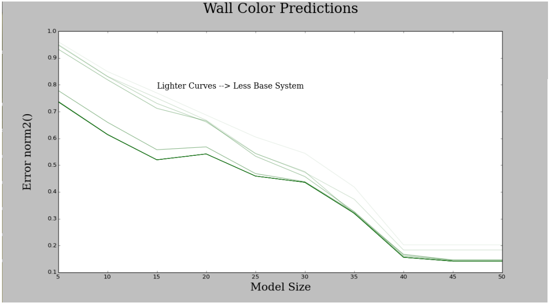


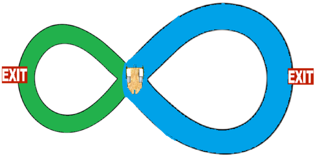


- $||f - \hat{f}|| = \sqrt{\sum_{x \in observations} (f(x) - f(\hat{x}))^2}$

Wall Color Predictions

We paint the first loop green and the second loop blue





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

Picking the Base System

- Picking operators to exploit structure

  Observations: $\{"a^{30}":10, "a^{60}":5, "b^{18}":15\}$

  Desired Base System: $A_{a^{30}}, A_{b^{18}}, A_a, A_b$
- Substring properties: **long**, **frequent**, **diverse**
- Solution: iterative greedy heuristic

Computing with the Base System

- Using the Base System well involves requires good **string partitions**
  Query string: "abcacb", Base System $= \{A_{ab}, A_{bca}, A_{cb}, A_a, A_b\}$
  Desired partition: "a—bca—cb""
- Goal: minimize matrices used
- Solution: dynamic programming

Conclusion and Future Work

- What's left for the Base System?

  1) Theoretical analysis

  2) Test heuristics on labyrinths

  3) Further optimize heuristics

Questions? Comments?