Learning and Discovery in Dynamical Systems with Hidden State
Supervised by Prakash Panangaden, Joelle Pineau, and Doina Precup

Milena Scaccia
mscacc@cs.mcgill.ca
Joint work with Dorna Kashef

August 30, 2007
Introduction

Learning and planning under uncertainty is one of the main aspects of modern AI research.

In the field of robotics:

- Robots must cope with environments that are partially observable, stochastic, dynamic and continuous.
- Partial observability, stochasticity are due to dealing with a large, complex world.
Introduction (cont’d)

- What is a good internal representation of the environment?
- What is the role of “state”? How do we pick a good state representation?
  - Some claim that “state” is merely a sufficient statistic for predicting the future
  - Robotics uses a standard notion of state, e.g., the x,y position of the robot at any given time, the angle of the joints, etc.
    - Obvious choice, but is it a good choice?
Problem Definition

- We want to learn the internal representation of a partially observable system by interacting with it.
- This is difficult because the state is hidden to an outside observer.
- There are observables that partly identify the state.
- We want to learn a model that will enable us to make good predictions on future action effects.
Partially Observable Systems

- Deterministic Kripke Automaton (DKA) 
\[ \mathcal{K} = \{ S, A, O, \delta : S \times A \rightarrow S, \gamma : S \rightarrow 2^O \} \]
Partially Observable Systems

- Deterministic Automaton with Stochastic Observations (DASO) $\mathcal{J} = (S, A, O, \delta : S \times A \rightarrow S, \gamma : S \times O \rightarrow [0, 1])$
Partially Observable Systems

- Partially Observable Probabilistic Automaton (POPA)
  \[ \mathcal{H} = (S, A, O, \tau : S \times A \times S \rightarrow [0, 1], \gamma : S \times O \rightarrow [0, 1]) \]
Learning and Discovery in Dynamical Systems with Hidden State

Goals

Our Goals

- **Long-term:** We want to find an algorithm that learns an automaton that best represents the data generated by the underlying system.

- **Short-term:** Compare and contrast properties of two candidate algorithms
  1. Isbell and Holmes’ Looping PST Algorithm (ICML 2006)
  2. Our Merge-Split Algorithm (derived from Gavalda, Keller, Pineau, and Precup’s (GKPP) PAC-Learning Algorithm (ECML 2006))
Isbell and Holmes present an solution for inferring hidden state in deterministic environments.

Isbell and Holmes claim that

1. hidden state can be fully represented by select pieces of history
2. a history-based hidden state representation can be made finite by excising uninformative regions of history.
   - E.g. Self-loops can fill up the history with uninformative action/observation steps.
Looping Prediction Suffix Tree Algorithm (cont’d)

**Figure:** The flip automaton in Mealy form

**Figure:** The looping PST for the flip automaton
Discussion

What have we learned?
- Unclear how to make predictions with looping PST

**Figure:** The flip automaton in Mealy form

**Figure:** The looping PST for the flip automaton

- The history $L_1 U_0 L_1$ is not valid in the flip automaton, but seems to be accepted by the automaton's looping PST.
GKPP Algorithm

- Given sequences of observations $o_1, ... o_n$, GKPP constructs an automaton which can generate this data with high probability.
- The algorithm maintains a list of “safe states” and “candidate states.”
GKPP Algorithm

- Given sequences of observations $o_1, \ldots, o_n$, GKPP constructs an automaton which can generate this data with high probability.
- The algorithm maintains a list of “safe states” and “candidate states”
  1. Begin with a single safe state, $s$
GKPP Algorithm

- Given sequences of observations $o_1, \ldots, o_n$, GKPP constructs an automaton which can generate this data with high probability
- The algorithm maintains a list of “safe states” and “candidate states”
  1. Begin with a single safe state, $s$
  2. In the induction step, we can either
GKPP Algorithm

- Given sequences of observations $o_1, \ldots, o_n$, GKPP constructs an automaton which can generate this data with high probability.
- The algorithm maintains a list of “safe states” and “candidate states”
  1. Begin with a single safe state, $s$
  2. In the induction step, we can either
     - Promote a candidate state to a safe state
Given sequences of observations $o_1, ... o_n$, GKPP constructs an automaton which can generate this data with high probability.

The algorithm maintains a list of “safe states” and “candidate states”:

1. Begin with a single safe state, $s$
2. In the induction step, we can either
   - Promote a candidate state to a safe state
   - Merge it with an existing safe state
GKPP Algorithm

- Given sequences of observations $o_1, \ldots, o_n$, GKPP constructs an automaton which can generate this data with high probability.
- The algorithm maintains a list of “safe states” and “candidate states”
  1. Begin with a single safe state, $s$
  2. In the induction step, we can either
     - Promote a candidate state to a safe state
     - Merge it with an existing safe state
     - Retain it as a candidate state
The Merge-Split Algorithm

- Merge-Split uses the same concept as the GKPP algorithm: it uses both state splitting and merging operations.
- Unlike GKPP, we have actions and observations.
- It works in deterministic environments.

- Two nodes are merged if their sets of possible future transitions are identical.
- If a node cannot be merged with any other existing node, it will be further expanded (split).
Merge-Split (cont’d)

**Figure:** The flip automaton in Mealy form

**Figure:** The merge-split automaton
Merge-Split (cont’d)

**Figure:** The flip automaton in Mealy form

**Figure:** The merge-split automaton
Neither of the two algorithms learn “minimal” model.

We had hypothesized that Merge-Split will always learn a smaller model than Looping PST, but this is false! Consider the seven-state float/reset automaton:
Looping PST learns a 25-state automaton
Merge-Split learns a 28-state automaton
Discussion (cont’d)

- Merge-Split should extend more easily to the probabilistic case than Looping PST
  - In the GKPP algorithm, notion of distinguishability parameter is specific to probability distribution
  - We are generalizing this for case of actions
Open Questions

- How can we generalize the Looping PST algorithm to work in probabilistic environments?

- How can we refine the two algorithms to learn the minimal automaton?

- Can we connect this with the notion of the double-dual representation? (Hundt et al. 2006)
  - The double-dual representation is the minimal version of the original machine with deterministic transition structure and no hidden state
  - This representation holds the promise of better learning and planning algorithms
  - How can we learn this representation from data?
References

