# Looping Suffix Tree-Based Inference of Partially Observable Hidden State

**Michael P. Holmes** MPH@CC.GATECH.EDU
**Charles Lee Isbell, Jr.** ISBELL@CC.GATECH.EDU
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280

## Abstract

We present a solution for inferring hidden state from sensorimotor experience when the environment takes the form of a POMDP with deterministic transition and observation functions. Such environments can appear to be arbitrarily complex and non-deterministic on the surface, but are actually deterministic with respect to the unobserved underlying state. We show that there always exists a finite history-based representation that fully captures the unobserved world state, allowing for perfect prediction of action effects. This representation takes the form of a looping prediction suffix tree (PST). We derive a sound and complete algorithm for learning a looping PST from a sufficient sample of sensorimotor experience. We also give empirical illustrations of the advantages conferred by this approach, and characterize the approximations to the looping PST that are made by existing algorithms such as Variable Length Markov Models, Utile Suffix Memory and Causal State Splitting Reconstruction.

## 1. Introduction

Partial observability is a longstanding problem for learning techniques that seek to model their environment, as the large POMDP and HMM literatures attest. Diverse methods for finding and using hidden state have been formulated, with goals such as time-series prediction, learning Markovian state for reinforcement learning, and uncertainty reduction for classical-style planning (Shalizi & Shalizi, 2004; McCallum, 1995; James & Singh, 2005; Holmes & Isbell, Jr., 2005). No general solution has yet emerged.

In this work we address the case of POMDPs in which the underlying transition and observation functions are deterministic. Such environments can appear to be arbitrarily complex and non-deterministic on the surface, but with knowledge of the underlying state their behavior becomes completely predictable and therefore controllable. We give a novel characterization of the hidden state learning problem that highlights two fundamental issues: 1) hidden state can be fully represented by select pieces of history, and 2) a history-based hidden state representation can be made finite by excising uninformative regions of history.

This characterization leads to a learning algorithm based on looping prediction suffix trees (PSTs). We show the algorithm to be sound and complete, and analyze several previous hidden state algorithms in terms of looping PSTs, including Variable Length Markov Models, Utile Suffix Memory, Causal State Splitting Reconstruction, and Predictive State Representations. We give empirical illustrations of the compromises made by some of these algorithms with respect to looping PSTs, finding them to be close but not exact approximations. Finally, we briefly discuss how the analysis presented here provides insight into the non-deterministic case.

## 2. Problem Setup

### 2.1. Definition

Adapting from (Rivest & Schapire, 1994), we define an environment in the class under consideration to be a tuple $E = (S, A, O, \delta, \gamma)$, where $S$ is the finite set of underlying world states, $A$ is the set of actions that can be taken from any state, $O$ is the set of possible observations, $\delta$ is a deterministic transition function $\delta(S, A) \rightarrow S$, and $\gamma$ is a deterministic (Mealy) observation function $\gamma(S, A) \rightarrow O$ generating an observation for each action taken. We also assume the

Appearing in *Proceedings of the $23^{rd}$ International Conference on Machine Learning*, Pittsburgh, PA, 2006. Copyright 2006 by the author(s)/owner(s).

world states to be strongly connected.[1] Environments under this definition form a subclass of POMDPs in which the state space is finite and the transition and observation functions are deterministic. Despite their fundamental determinism, such environments can be highly complex and non-deterministic at the level of action and observation.

Some additional notation we will employ:

- $\gamma(s_i, a_1 \ldots a_n)$ denotes the observation sequence $o_1 \ldots o_n$ generated by taking action sequence $a_1 \ldots a_n$ from state $s_i$.
- A history $h$ is a sequence of actions and observations $a_1 o_1 \ldots a_n o_n$, beginning with an action and ending with an observation. If $p$ and $h$ are two histories, writing them sequentially $(ph)$ refers to their concatenation.
- $S^\lambda = S \cup \lambda$, where $\lambda$ is a special null state.
- $h(S^\lambda) : S^\lambda \mapsto S^\lambda$ is a function mapping each state $s_i \in S$ to the state reached by history $h$ starting in $s_i$, or to the null state $\lambda$ if $h$ is impossible from $s_i$; $\lambda$ is always mapped to itself.
- $trans(h) = \{ao : a \in A,\ o \in O, \text{ and } ao \text{ is a possible transition following } h\}$.

Our goal is to build a sound and complete predictive model of the environment given only a history of sensorimotor experience. By sound and complete, we mean that the model correctly predicts the observation that will follow any action, given the history preceding the action. We first present a theoretical analysis showing that every environment has a finite history suffix tree-based representation that allows for perfect prediction. We then derive an algorithm for learning such a representation from sensorimotor experience, characterize various existing hidden state algorithms in terms of our solution, and give empirical performance comparisons that show the approximations being made to the complete solution.

## 2.2. Prediction Suffix Trees

The only source of information about hidden state is the learner's history of interaction with the environment. Yet not all history is informative at all times: the fact that you drank a glass of water ten years ago probably has no bearing on whether your car will start today. We would like to track some minimal, finite

[1]If not, the learner eventually falls into a strongly connected component that then becomes the whole accessible environment (Rivest & Schapire, 1994).

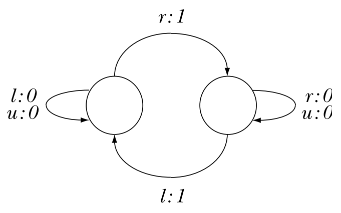

*Figure 1.* The **flip** automaton in Mealy form.

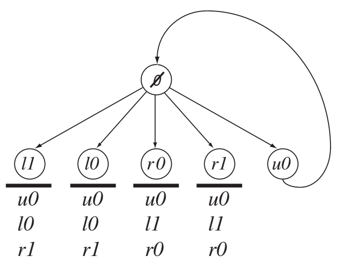

*Figure 2.* A looping prediction suffix tree for the **flip** automaton. Bars under leaf nodes indicate expansion termination, with each node's transitions listed beneath the bar. Note that the $u0$ node loops back to the root.

amount of history sufficient to predict the future. In short, we need a sufficient statistic, as discussed in (Shalizi & Shalizi, 2004). As we have defined the problem, this means using history to infer the current state. Figure 1 contains a simple two-state environment illustrating the fact that each state can only be reached by particular sensorimotor histories. For example, $l1$ enters the left state but not the right state, so after seeing $l1$ we are definitely in the left state. Based on this idea, we seek to represent the world state using observation-terminated history suffixes, i.e., the end portion of the action/observation history $a_1 o_1 \ldots a_n o_n$ that led to the current observation. In general this will require more than just the most recent action/observation step, and each additional step can help filter out possible end states. For instance, in Figure 1, $u0$ leads to both the left-hand and right-hand states, while $r1u0$ leads only to the right-hand state.

As shown in Figure 2, it is natural to represent the set of history suffixes as a tree (Ron et al., 1994). In a suffix tree, any sufficiently long history can be sorted down to a leaf node by following the branches labeled by its suffix in reverse order. With the addition of transition predictions at the leaves, the tree becomes a prediction suffix tree (PST). If each leaf node suffix corresponds to a unique state, its transition predictions will be deterministic. Our goal is to learn a PST that maps any history to a leaf that accurately predicts its future transitions.

## 3. Finite Representability of Hidden State

In learning a predictive model we can assume without loss of generality that the environment is in a minimized form, with minimality defined as follows:

**Definition.** An environment $E$ is minimized iff, for any two states $s_i$ and $s_j$, there exists a distinguishing sequence of actions $d_{ij} = a_1 \ldots a_n$ such that $\gamma(s_i, d_{ij}) \neq \gamma(s_j, d_{ij})$.

The distinguishing sequence $d_{ij}$ provides a test that can be executed to rule out one or both of $s_i$ and $s_j$ as possible start states. It does not, however, uniquely determine the start or end states. We will use the term *resolving history* to refer to a sequence of alternating actions and observations that uniquely determines the end state:

**Definition.** A history sequence $h$ *resolves* to state $s_i$ iff $h(S^\lambda)$ maps every state in $S^\lambda$ either to $\lambda$ or to $s_i$, with at least one state mapping to $s_i$.

Thus, immediately following a history $h$ that resolves to $s_i$, the environment must be in state $s_i$. The environment is therefore Markovian relative to any resolving history. In general, an arbitrary history $h$ may map some states to $\lambda$, and any prefix $p$ may also map some states to $\lambda$. Prepending the prefix gives the overall functional mapping $h(p(S^\lambda))$. Because all histories map $\lambda$ to $\lambda$, anything mapped to $\lambda$ by $p$ must also be mapped to $\lambda$ by $ph$. This may include states that $h$ alone does not map to $\lambda$. Thus, backward extension of a history acts as a filtering mechanism that can never decrease the number of $\lambda$ mappings, but can certainly increase them. The process of resolving is essentially the process of increasing the number of $\lambda$ mappings until the only remaining non-$\lambda$ mappings all lead to the same state.

A resolving sequence is distinct from a *homing sequence*, a term used in the literature to refer to a sequence composed purely of actions such that, when executed, the resultant observation sequence uniquely identifies the end state. A homing sequence represents a set of resolving sequences, one for each possible observation sequence generated by its actions.

It is known that every finite, deterministic environment has at least one homing sequence (Shen, 1993b). This, in turn, implies the existence of at least one resolving sequence. Given a sequence $h$ that resolves to $s_i$, we can construct a new resolving sequence by adding any action/observation step $ao$ that $s_i$ can generate. Because the environment is deterministic, the new sequence $hao$ must resolve to $s_j = \delta(s_i, a)$. By construction, there must exist infinitely many resolving sequences that can be obtained through such extensions. Further, because the environment is strongly connected, there exist infinitely many resolving sequences that end in any particular state $s_i$.

If all resolving sequences for each state were known, they could be matched against sensorimotor history to infer the current state at any time; however, some of these sequences are longer than necessary. For example, a resolving sequence $h$ could have an arbitrary (but possible) sequence $p$ prepended to it, and the resultant sequence $ph$ would still resolve to the same state. Clearly we would prefer resolving sequences that are as short as possible. This leads us to define the set of *minimal resolving sequences*:

**Definition.** The set of *minimal resolving sequences* for a state $s_i$ is the set of resolving sequences $h$ such that no sequence $h'$ formed by removing steps from the left side of $h$ is also resolving.

Thus, in the $ph$ case, because removing $p$ to form $h$ still gives a resolving sequence, $ph$ is not minimal resolving. Note, however, that the minimal resolving sequences can have unbounded length. This is because, while some portions of a resolving sequence are informative in that they increase the number of $\lambda$ mappings, other portions may not be informative at all, i.e. they merely lengthen the history without increasing the $\lambda$ mappings. A minimal resolving history can potentially repeat uninformative subsequences arbitrarily many times before finally adding the informative portion that terminates the expansion. We will refer to uninformative sequences as *excisable*, and to informative sequences as *non-excisable.*

**Definition.** For any two histories $h$ and $q$ such that $h = eq$, the sequence $e$ is *excisable* from $h$ iff $\forall p \; trans(ph) = trans(pq)$. Otherwise $e$ is non-excisable.

This definition gives us an observational criterion for deciding the excisability of a history subsequence. We now give a sufficient condition for excisability in terms of equivalent histories:

**Definition.** For two histories $h$ and $q$ such that $h = eq$, we say that $h$ and $q$ are functionally equivalent if $h(S^\lambda) = q(S^\lambda)$, i.e. the mappings induced by the two histories are identical.

**Lemma 1.** *For two histories $h$ and $q$ such that $h = eq$, if $h$ and $q$ are functionally equivalent then $e$ is excisable from $h$.*

*Proof.* If $h$ and $q$ are functionally equivalent, we have, for any prefix history $p$ and any state $s$, $ph(s) =$

$h(p(s)) = q(p(s)) = pq(s)$. Therefore $ph$ and $pq$ are also functionally equivalent, and, because they each leave the same set of possible (non-$\lambda$) end states, we have $trans(ph) = trans(pq)$. Because this holds for all $p$, we have satisfied the definition of excisability and conclude that $e$ is excisable from $h$. □

Excisability is useful because it facilitates a finite representation of the minimal resolving sequences. Consider the tree of all possible history suffixes. If we have a sequence $h = eq$ and $e$ is excisable from $h$, then for all predictive and state-resolving purposes we can treat instances of $h$ as though they were actually $q$. Thus, if in sorting our history down the suffix tree we reach node $h$, we might just as well loop back up to $q$ as continue sorting down from $h$—all transition predictions will remain the same (see Figure 2). This is the basis for *looping* predictive suffix trees. Whereas a suffix tree representing the minimally resolving sequences could be infinite in depth, we now show that a looping suffix tree that loops back on excisable subsequences is always finite and retains full predictive power.

**Lemma 2.** *Every branch of a history suffix tree either becomes resolving or reaches a level that begins with an excisable sequence after finite depth.*

*Proof.* Some branches will resolve at finite depth because resolving sequences exist, as previously noted. Consider, then, the remaining branches, which must remain unresolving to infinite depth. For a given branch representing the infinite history $h$, let $h_i$ be the history represented by the suffix starting at the root and extending down to level $i$. Each $h_i$ represents a particular function mapping $S^\lambda$ onto $S^\lambda$. But there are only finitely many such functions, namely $|S^\lambda|^2 = |S|^2 + 2|S| + 1$. Therefore, after no more than $|S|^2 + 2|S| + 1$ levels, there will be a functional repeat such that $h_j$ is functionally equivalent to $h_k$, $k > j$, $k, j < |S|^2 + 2|S| + 1$. Because $h_k = eh_j$, we can invoke Lemma 1 to conclude that $e$ is excisable from $h_k$ and therefore $h$ becomes excisable after finite depth $k$. □

**Theorem 1.** *Any finite, deterministic POMDP can be perfectly predicted by a finite looping PST, given a resolving history to start with.*

*Proof.* First, it is clear that without a resolving history perfect prediction cannot be made, as the current state will not have been resolved. So the best we can hope for is to predict correctly given a resolving history. We therefore need to show that there is a finite, looping PST that correctly captures all predictions of resolving histories.

By Lemma 2, we know that each branch of the full suffix tree either resolves or reaches a level where it begins with an excisable sequence after finite depth. If a branch $h$ resolves at finite depth, then we cut it off there, forming a leaf node in which we store $trans(h)$ as the predictions, which will be deterministic and correct.

For a branch that does not resolve at finite depth, let $h_k$ be the first suffix along the branch that begins with an excisable sequence $e$ such that $eh_j = h_k$. We know that $h_k$ will be at finite depth, and we place a loop from $h_k$ to $h_j$. No expansion of the tree below $h_k$ takes place. A history being sorted down through $h_k$ will loop back to $h_j$ and continue to be sorted down as though it had reached $h_j$ rather than $h_k$. Because, by definition of excisability, $\forall p\ trans(ph_k) = trans(ph_j)$, any resolving $ph_k$ will be sorted down to $ph_j$, which will have the same deterministic transitions and therefore make the correct predictions. Thus, if we loop on all excisable sequences we will make their branches finite and still make all predictions correctly.

We have shown by construction the existence of a looping PST that gives correct predictions for any resolving history and whose branches all terminate after finite depth, and this establishes the theorem. □

Looping provides Kleene star-like representational power: if $h = eq$ loops to $q$, $q$'s node no longer represents the suffix $q$ alone, but instead represents the infinite set $e^*q$. Arbitrarily long repetitions of uninformative sequences are excised. This solves a classic problem with history-based approaches wherein states with self loops can fill up the history with so many uninformative action/observation steps that any finite window of history can be rendered useless. Not only does PST looping allow the infinite set of resolving histories to be mapped onto a finite representation, it also gives the power to generalize: a tree with loops gains the ability to correctly predict the transitions of suffixes it has never seen.

## 4. Algorithmic Results

The preceding analysis suggests an algorithm for learning a PST through interaction with the environment: just follow the construction of the looping PST as given in the proof, expanding branches until they resolve or become loopable. But first we must establish the ability to discern the branch termination and looping criteria from data. We accomplish this by way of a definition and two lemmas.

**Definition.** We say that a history is *sufficient* if every action has been taken often enough and in a sufficient number of contexts so as to reveal the full transition set $trans(h)$ for every node $h$ considered by a looping PST building algorithm.

Correct decisions could certainly be made by an algorithm having access to an oracle telling it what transitions are possible from each node. A fully representative history is equivalent to such an oracle, and need only be finitely long for a finite number of node expansions. Such a history is, in reality, exponentially unlikely if actions are chosen randomly (hence the NP-completeness result in the case of passive learning (Freund et al., 1993)). Strategic exploration can produce a sufficient history more efficiently, but a discussion of exploration strategies is beyond the scope of this paper.

**Lemma 3.** *Given a sufficient history, correct prediction on any non-looping branch $h$ is obtained by terminating at the first level $h_i$ such that $trans(h_i)$ is deterministic for each action.*

*Proof.* If $trans(h_i)$ is deterministic for each action, which only a sufficient history can reveal for certain, then $h_i$ must map $S^\lambda$ onto a subset $R_{h_i}$ of states that all have identical single-step action/observation transitions. If we extend $h_i$ by $p$ to get the resolving sequence $h = ph_i$, the $h_i$ part of $h$ performs the same mapping but on the subset $R_p \subseteq S^\lambda$ left over after $p$. Thus, $R_{ph_i} \subseteq R_{h_i}$, and therefore $trans(h) = trans(h_i)$, so the correct predictions can be made at $h_i$ without expanding all the way to $h$. □

Because we cannot see the underlying state to tell when a branch has resolved, Lemma 3 is crucial as an observable termination criterion. Transitions are tabulated from data, so as long as we have a sufficient sample of experience to tell us the full transition set of any history we encounter in our tree expansion, we can apply Lemma 3 to decide whether to terminate. Without a sufficient history, this criterion may lead to early termination, but it will at least be correct with respect to past experience and in the sufficient history limit will lead to proper convergence.

**Lemma 4.** *If, given a sufficient history, there is no prefix $p$ for which $trans(ph) \neq trans(pq)$ $(h = eq)$, then $e$ is excisable from $h$ and $h$ can therefore be looped to $q$.*

*Proof.* According to the definition of excisability, $e$ is excisable from $h$ if and only if $\forall p$ $trans(ph) = trans(pq)$. If $e$ is not excisable from $h$, we therefore have $\exists p$ $trans(ph) \neq trans(pq)$. Since we have a sufficient history, if there were such a $p$ it would appear often enough in the history to reveal that $trans(ph) \neq trans(pq)$. But since we are given that no such inequality exists, we can conclude that $trans(ph)$ and $trans(pq)$ are equal, and therefore $e$ is excisable from $h$ and $h$ can be safely looped to $q$. □

The utility of Lemma 4 is that it allows us to assume we should place a loop unless some finite prefix contradicts it. With these lemmas in place, we now give a looping PST learning algorithm and show that it is sound and complete when given a sufficient history.

**Algorithm 1** Algorithm for learning a looping PST.

```
LLT($h$)
create root node $r$ representing the empty suffix
add $r$ to an expansion queue
while nodes remain in queue do
  remove the first node $x$
  find all transitions following instances of $x$ in $h$
  if $x$'s transition set is deterministic then
    continue
  else if isLoopable($x$) then
    create loop from $x$ to the highest to highest possible ancestor
  else
    for each one-step extension of $x$ in $h$ do
      create child node and add to queue
    end for
  end if
end while
return $r$
```

**Algorithm 2** Subroutine for determining loopability.

```
isLoopable($x$)
for each ancestor $q$ of $x$, starting at the root do
  for all prefixes $p$ such that $px$ or $pq$ occurs in $h$ do
    if $pq$ or $px$ does not appear in $h$ OR $trans(px) \neq trans(pq)$ then
      return False
    end if
  end for
end for
return True
```

**Theorem 2.** *When given a sufficient history, LLT returns a finite looping PST that correctly predicts the transitions following any resolving history.*

*Proof.* LLT follows the constructive proof of Theorem 1, but with empirical observable criteria for placing loops and terminating expansion. By Lemma 3, terminating upon reaching transition determinism is correct when based on a sufficient history. By Lemma 4, allowing loops except when some prefix shows transition disagreement between the looping and looped-to nodes is also correct if the history is sufficient. LLT with a sufficient history therefore constructs a looping PST that will correctly predict the transitions of any resolving history. □

### 4.1. Practical Considerations

In practice, it can be difficult to obtain a sufficient history. This is especially problematic for loop creation. The looping criterion requires that for $h$ to loop to $q$ ($h = eq$) there be no prefix $p$ for which $trans(ph) \neq trans(pq)$. If the history is not sufficient, $trans(ph)$ and $trans(pq)$ can be appear unequal due to the fact that not all actions have been taken from $ph$ or $pq$. As $p$ becomes longer, the probability of missing transitions increases exponentially, with a net result of many valid loops being rejected, yielding overly large trees and poor generalization.

We thus substitute Lemma 4 with an alternative criterion based on prediction accuracy: a loop is allowed in the tree if it does not decrease prediction accuracy over the training data. This frees us from the requirement of seeing every supporting transition before we can loop. In the limit of a sufficient history, a false loop would decrease prediction accuracy, so the accuracy-based criterion becomes equivalent to Lemma 4. This modification is handled with an alternate version of the **isLoopable** subroutine, which we give as Algorithm 3.

**Algorithm 3** Alternate subroutine for determining loopability.

```
isLoopable2($x$)
for each ancestor $q$ of $x$, starting at the root do
    simulate loop from $x$ to $q$
    evaluate prediction accuracy over past history
    if accuracy does not decrease then
        return True
    end if
end for
return False
```

## 5. Analysis of Related Approaches

Looping PSTs provide a framework in which to understand and compare existing approaches to partial observability. In this section we analyze prior work that deals with hidden state. Even though most of these algorithms were designed for non-deterministic environments, our analysis of the way they handle environments with underlying determinism clarifies some of the fundamental choices made by each.

As we have seen, the states of a deterministic finite environment can be inferred by selectively tracking finite pieces of history. The main issue in any algorithm dealing with partial observability is how to map the infinite set of possible histories to some finite representation, and this is the focus of our analysis.

### 5.1. CSSR, VLMMs, and USM

Causal State Splitting Reconstruction (CSSR; Shalizi & Shalizi, 2004) constructs time-series predictors by exhaustively expanding a history suffix tree to a user-specified maximum depth $L_{max}$. It includes a method for aggregating the expanded suffixes that belong to the same "causal state," but this does not increase its predictive power beyond that provided by the fixed-depth exhaustive suffix tree expansion. Excisable history is not explicitly handled, but fixing the depth of the tree can be seen as an approximation that prevents unbounded expansion down excisable branches. Note that, because the expansion is exhaustive, even minimal resolving suffixes will be expanded if their length is less than $L_{max}$.

Variable Length Markov Models (VLMMs) also expand history suffix trees in the hope of producing a Markovian representation of state. Unlike CSSR, VLMM trees are variable in depth and employ an empirical stopping criterion. The criterion of (Ron et al., 1994) is representative: a child suffix node is created only if 1) the probability of its suffix occurring is above some threshold, and 2) the transition behavior of the child differs significantly (by a statistical test such as KL divergence) from that of the parent. This criterion, unlike that of CSSR, will stop at a minimal resolving suffix, giving rise to a variable-depth tree. Again, however, excisable sequences are not explicitly dealt with, and the expansion stopping criterion can be seen as an alternative approximation to prevent unbounded expansion of excisables.

The Utile Suffix Memory (USM) algorithm (McCallum, 1995) is very similar to VLMM learning, but with a reward-based stopping criterion that expands a child node only if the distribution of next-step reward appears to differ statistically significantly from that of its parent. This is yet another approximation to prevent unbounded expansion of the tree instead of directly handling excisable sequences. UTree (McCallum, 1995) is an extension to USM that allows the observations to be broken up into individual sensors and added to the suffix tree one at a time.

## 5.2. Schema Learning, PSRs, etc.

Schema Learning (Holmes & Isbell, Jr., 2005; Drescher, 1991) builds STRIPS-like units called schemas that predict the effects of taking actions under particular preconditions. Like UTree, schemas deal with sensor vectors rather than monolithic observations. As a predictive unit, a schema is similar to a PST branch of depth one; however, schema preconditions can include history information in the form of *synthetic items*. These items are created by the learner and essentially act as history bits indicating that one of several schemas was activated on the previous timestep. Because synthetic items are incorporated into schemas' preconditions, they add depth to the suffix branch being represented. If the schemas activating a given synthetic item have additional synthetic items in their preconditions, this effectively adds an additional step to the history being captured, and so on to arbitrary depth. Further, sharing of a synthetic item among multiple schemas' preconditions effectively produces loops between the suffix branches, giving schema learning a representational power equivalent to looping PSTs, though this is not fully exploited by current algorithms. Although space does not permit elaboration, the LLT algorithm suggests systematizations that could improve schema learning's use of synthetic items.

Predictive State Representations (PSRs; Littman et al., 2002) represent state as a finite set of prediction probabilities over multi-step future action/observation sequences called tests. The tests remain fixed while their probabilities are updated at each timestep. Let $P_i$ be the set of probabilities at timestep $i$; the update is of the functional form $P_{i+1} = f(P_i, a_i o_i)$, where $a_i o_i$ is the action/observation pair seen at timestep $i$. Recursively expanding this expression shows that $P_i$ is actually a function of the entire history, i.e., histories of arbitrary length are mapped onto a finite representation as in looping PSTs. The intent is for $P_i$ to contain predictions for enough tests so as to form a sufficient statistic (which, in the deterministic case, means $P_i$ resolves the current state). One way to achieve sufficiency is for $P_i$ to contain predictions for all one-step tests. This means that, for deterministic environments, looping PSTs *are* PSRs, because looping PST leaf nodes contain all one-step predictions. In other words, LLT constitutes a PSR discovery algorithm for deterministic environments. This is significant because PSR discovery in general remains an open problem.

Other algorithms of note include Shen's D* (Shen, 1993b; Shen, 1993a) and the diversity-based automaton inference of Rivest and Schapire (Rivest & Schapire, 1994); unfortunately, space does not permit analysis of these here.

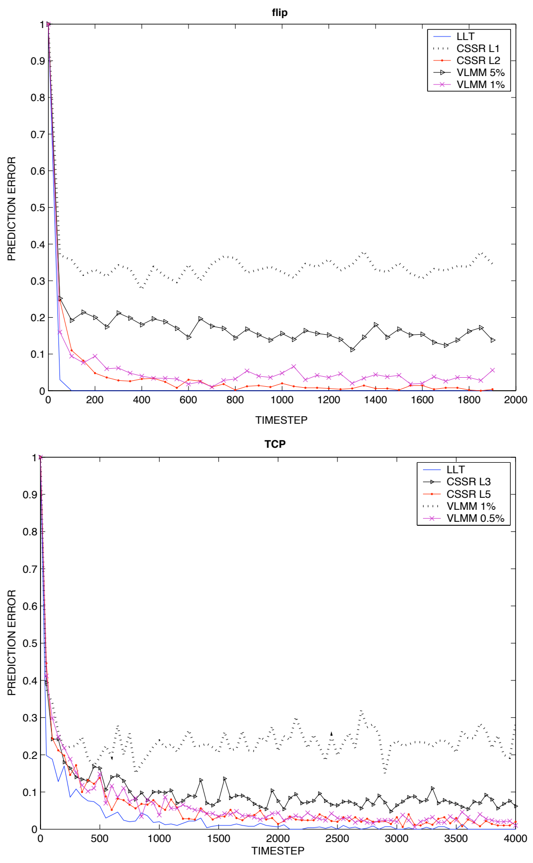

*Figure 3.* **Comparative performance.** (top) Results for the **flip** environment. (bottom) Results for the TCP automaton. The algorithms were run from scratch after every 50 timesteps, and their output was used to predict the next 50 timesteps. Each data point represents the average prediction error for 50 timesteps over 10 runs with actions chosen at random.

## 5.3. Empirical Comparison

As an illustration of the difference made by using looping PSTs, we present a comparison between the suffix tree expansion performance of the LLT, CSSR, and VLMM criteria. Note that these are simple experiments in synthetic environments intended only to highlight the tradeoffs involved. We ran LLT with the looping criterion described in section 4.1, then ran a non-looping suffix tree expander with a fixed depth criterion (CSSR) and a probability-of-occurrence criterion (VLMM). The CSSR and VLMM criteria were run against LLT on the **flip** automaton and on a 10-state, 3-observation automaton based on the server-side portion of the TCP network protocol. Results are shown in Figure 3.

In both cases, LLT produces a correct model with zero prediction error. In **flip**, neither CSSR nor VLMM can reach a correct model because, no matter how deep they go, $(u0)^*$ bunches can occur in arbitrary lengths and confound them, whereas LLT excises all $u0$ steps. Obviously CSSR could have its $L_{max}$ increased and VLMM could have its threshold probability dropped to such a point that the probability of encountering a $(uo)^*$ sequence longer than the tree depth would approach zero; however, such a model would still be imperfect and would be much larger than the looping PST version.

In the TCP case, the situation is much the same, but because of the more complex dynamics there are deeper trees and longer learning times. CSSR and VLMM do approach (but never reach) zero error, but not as quickly as LLT because of the generalization provided by looping. The LLT stopping and looping criteria lead to the fewest node expansions, while VLMM produces more nodes because it expands loops, and CSSR's exhaustive expansion produces the most nodes of all.

## 6. Discussion and Future Work

We have shown that hidden state in deterministic finite POMDPs can be fully represented, without loss of predictive power, by finite looping PSTs. This characterization has yielded the LLT algorithm for learning looping PSTs, and we have shown it to be sound and complete when given a sufficient history to learn from. We have also shown how the LLT algorithm can be used to characterize the workings, approximations, and limitations of existing hidden state algorithms, at least with respect to their handling of the class of environments treated here. LLT outperforms some of them on simple, illustrative problems.

It is our hope that this work will provide a foundation for characterizing environments with non-deterministic hidden state as well. We are currently exploring this direction and are finding the notions developed for the deterministic case to have clear analogues in the non-deterministic case. For example, resolving histories generalize to *almost-certainly approximately resolving* histories that determine the current belief state and serve as approximate sufficient statistics. Though these sequences can require infinite length for full belief state resolution, specification of an error tolerance appears to allow a finite representation to emerge. Ultimately the intent is that more informed and effective methods of learning state representations will emerge from a clearer theory of history-based hidden state extraction.

## Acknowledgments

We acknowledge the support of DARPA under contract No. HR0011-04-1-0049.

## References

Drescher, G. (1991). *Made-up minds: a constructivist approach to artificial intelligence.* MIT Press.

Freund, Y., Kearns, M., Ron, D., Rubinfeld, R., Schapire, R. E., & Sellie, L. (1993). Efficient learning of typical finite automata from random walks. *25th ACM Symposium on the Theory of Computing* (pp. 315–324).

Holmes, M. P., & Isbell, Jr., C. L. (2005). Schema learning: Experience-based construction of predictive action models. *Advances in Neural Information Processing Systems* (pp. 585–592). MIT Press.

James, M. R., & Singh, S. (2005). Planning in models that combine memory with predictive state representations. *Proceedings of the Twentieth National Conference on Artificial Intelligence* (pp. 987–992). AAAI Press.

McCallum, A. W. (1995). *Reinforcement learning with selective perception and hidden state.* Doctoral dissertation, University of Rochester.

Rivest, R. L., & Schapire, R. E. (1994). Diversity-based inference of finite automata. *Journal of the ACM*, *41*, 555–589.

Ron, D., Singer, Y., & Tishby, N. (1994). The power of amnesia. *Advances in Neural Information Processing Systems* (pp. 176–183). Morgan Kaufmann.

Shalizi, C. R., & Shalizi, K. L. (2004). Blind construction of optimal nonlinear recursive predictors for discrete sequences. *Uncertainty in Artificial Intelligence* (pp. 504–511). AUAI Press.

Shen, W. M. (1993a). Discovery as autonomous learning from the environment. *Machine Learning*, *12*, 143–165.

Shen, W. M. (1993b). Learning finite state automata using local distinguishing experiments. *International Joint Conference on Artificial Intelligence* (pp. 1088–1093). Morgan Kaufmann.