Learning Partially Observable Deterministic Action Models

Eyal Amir
Computer Science Department
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
eyal@cs.uiuc.edu

Abstract

We present the first tractable, *exact* solution for the problem of identifying actions' effects in partially observable STRIPS domains. Our algorithms resemble Version Spaces and Logical Filtering, and they identify all the models that are consistent with observations. They apply in other deterministic domains (e.g., with conditional effects), but are inexact (may return false positives) or inefficient (we could not bound the representation size). Our experiments verify the theoretical guarantees, and show that we learn STRIPS actions efficiently, with time that is significantly better than approaches for HMMs and Reinforcement Learning (which are inexact). Our results are especially surprising because of the inherent intractability of the general deterministic case. These results have been applied to an autonomous agent in a virtual world, facilitating decision making, diagnosis, and exploration.

1 Introduction

Autonomous agents have limited prior knowledge of their actions' preconditions and effects when they explore new domains. They can act intelligently if they learn how actions affect the world and use this knowledge to respond to their goals. This is important when their goals change because then they can reason about their actions instead of trying them in the world.

Learning actions' effects and preconditions is difficult in partially observable domains. The world state is not known completely, actions' effects mix with each other, and it is hard to associate change in one feature with a specific action or situation. Thus, it is not surprising that work so far has been limited to fully observable domains (e.g., [Wang, 1995; Pasula *et al.*, 2004]) and to hill-climbing (EM) approaches that have unbounded error in deterministic domains (e.g., [Ghahramani, 2001; Boyen *et al.*, 1999]).

This paper presents an approach called *SLAF* (Simultaneous Learning and Filtering) for exact learning of action's effects and preconditions in partially observable deterministic domains. This approach determines a set of possible transition relations, given a sequence of actions and partial observations. It is *online*, and updates a logical formula that models the possible transition relations and world states with every time step. It finds exactly those transition relations when actions are STRIPS (i.e., no conditional effects) or they map states 1:1.

Our algorithms take polynomial time in the number of features, n, and the number of time steps, T, for many cases. Their exact complexity varies with properties of the domain. For example, the overall time for learning STRIPS actions' effects is $O(T \cdot n)$. For other cases the update per time step takes linear time in the representation size. We can bound this size by $O(n^k)$ if we approximate the representation with a k-CNF formula, yielding an overall time of $O(T \cdot n^k)$ for the entire algorithm.

Our experiments verify these theoretical results and show that our algorithms are faster and better qualitatively than related approaches. For example, we can learn STRIPS actions' effects in domains of > 100 features *exactly*. In contrast, work on learning in Dynamic Bayesian Networks (e.g., [Boyen *et al.*, 1999]), reinforcement learning in POMDPs (e.g., [Littman, 1996]), and Inductive Logic Programming (ILP) (e.g., [Wang, 1995]) either approximate the solution with unbounded error for deterministic domains, or take time $\Omega(2^{2^n})$ (thus, are inapplicable in domains larger than 10 features). Section 7 provides a comparison with these and other works.

Our technical advance for deterministic domains is important for many applications such as automatic software interfaces, internet agents, virtual worlds, and games. Other applications, such as robotics, human-computer interfaces, and program and machine diagnosis can use deterministic models as approximations. Finally, understanding the deterministic case better can help us develop better results for stochastic domains.

In the following, Section 2 defines SLAF precisely,

Section 3 provides a deduction-based exact SLAF algorithm, Section 4 presents tractable model-update algorithms, Section 5 gives sufficient conditions and algorithms for keeping the model representation compact (thus, overall polynomial time), and Section 6 presents experimental results.

2 SLAF Semantics

We define our SLAF problem with the following formal tools, borrowing intuitions from work on Bayesian learning of Hidden Markov Models (HMMs) [Ghahramani, 2001] and Logical Filtering [Amir and Russell, 2003].

A *transition system* is a tuple $\langle \mathcal{P}, \mathcal{S}, \mathcal{A}, R \rangle$, where

- $\mathcal{P}$ is a finite set of propositional fluents;
- $\mathcal{S} \subseteq Pow(\mathcal{P})$ is the set of world states;
- $\mathcal{A}$ is a finite set of actions;
- $R \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition relation.

Thus, a *world state*, $s \in \mathcal{S}$, is a subset of $\mathcal{P}$ that contains propositions true in this state, and $R(s, a, s')$ means that state s' is a possible result of action a in state s. Our goal in this paper is to find R, given known $\mathcal{P}$, $\mathcal{S}$, $\mathcal{A}$, and a sequence of actions and partial observations (logical sentences on any subset of $\mathcal{P}$).

A *transition belief state* is a set of tuples $\langle s, R \rangle$ where s is a state and R a transition relation. Let $\mathfrak{R} = Pow(\mathcal{S} \times \mathcal{A} \times \mathcal{S})$ be the set of all possible transition relations on $\mathcal{S}, \mathcal{A}$. Let $\mathfrak{S} = \mathcal{S} \times \mathfrak{R}$. When we hold a transition belief state $\rho \subseteq \mathfrak{S}$ we consider every tuple $\langle s, R \rangle \in \rho$ possible.

For example, consider the situation presented in Figure 1. There, we have two rooms, a light bulb, a switch, an action of flipping the switch, and an observation, E (we are in the east room). The real states of the world, $s2, s2'$ (shown in the top part), are unknown to us.

The bottom part of Figure 1 demonstrates how our knowledge evolves after performing the action sw-on. ρ_1, ρ_2 are our respective transition belief states. In ρ_1 we know that the possible world states are $s2$, $s1$, or $s3$ ($s1, s3$ are arbitrary world states), with $R2$, $R1$, and $R3$, their respective transition relations. ρ_2 is the resulting transition belief state after action sw-on. Action sw-on takes state $s2$ to $s2'$ according to transition relation $R2$, so $\langle s2', R2 \rangle$ is a pair ρ_2. Similarly, it takes state $s1$ to one of $s1', s1''$ according to transition relation $R1$, and it takes $s3$ to $s3'$ according to $R3$. Thus, $\langle s1', R1 \rangle, \langle s1'', R1 \rangle, \langle s3', R3 \rangle$ are in pair ρ_2. Finally, observing E eliminates the pair $\langle s3', R3 \rangle$ from ρ_2, if E is false in $s3'$.

Definition 2.1 (SLAF Semantics) *Let $\rho \subseteq \mathfrak{S}$ be a transition belief state. The* SLAF *of ρ with actions and observations $\langle a_j, o_j \rangle_{1 \le j \le t}$ is defined by*

1. $SLAF[a](\rho) = \{\langle s', R \rangle \mid \langle s, a, s' \rangle \in R,\ \langle s, R \rangle \in \rho\}$;
2. $SLAF[o](\rho) = \{\langle s, R \rangle \in \rho \mid o \text{ is true in } s\}$;
3. $SLAF[\langle a_j, o_j \rangle_{i \le j \le t}](\rho) = SLAF[\langle a_j, o_j \rangle_{i < j \le t}](SLAF[o_i](SLAF[a_i](\rho)))$.

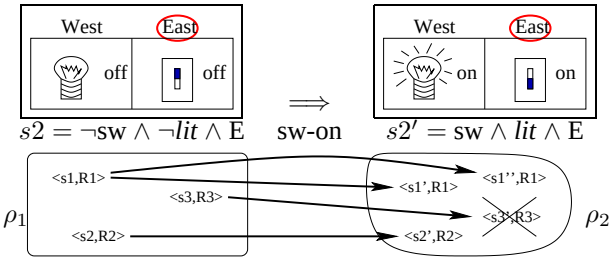

Figure 1: *Top*: Two rooms and flipping the light switch. *Bottom*: SLAF semantics; progressing an action (the arrows map state-transition pairs) and then filtering with an observation (crossing out some pairs).

Step 1 is progression with a, *and Step 2* filtering with o.

We assume that observations (and observation model relating observations to state fluents) are given to us as logical sentences over fluents after performing an action. They are denoted with o. When actions can be inexecutable or may fail, it is useful to assume that $\mathcal{P}$ contains a special boolean fluent, OK, whose value is the success of the last action attempted.

Transition belief state ρ generalizes version spaces (e.g., [Wang, 1995]) as follows: If the current state, s, is known, then the version space's lattice contains the set of transition relations $\rho^s = \{R \mid \langle s, R \rangle \in \rho\}$. It also generalizes belief states: If the transition relation, R, is known, then the belief state (set of possible states) is $\rho^R = \{s \mid \langle s, R \rangle \in \rho\}$ (read ρ *restricted to* R), and Logical Filtering [Amir and Russell, 2003] of belief state σ and action a is equal to (thus, we define it as)

$$Filter[a](\sigma) = (SLAF[a](\{\langle s, R \rangle \mid s \in \sigma\}))^R.$$

3 SLAF via Logical Inference

Learning transition models using Definition 2.1 directly is intractable because it takes space $\Omega(2^{2^{|\mathcal{P}|}})$ in many cases. Instead, in this section we represent transition belief states more compactly (sometimes) using logic, and compute SLAF using general purpose logical inference.

We represent every *deterministic* transition relation, R, with a language, L, of propositions a^F_G whose meaning is "If G holds, then F will hold after executing a". We have $a^F_G \in L$ for every action $a \in \mathcal{A}$, F a literal (possibly negated proposition) from $\mathcal{P}$, and G a complete term over $\mathcal{P}$ (a conjunction of literals from $\mathcal{P}$ such that every fluent appears exactly once). We call $\mathcal{F}, \mathcal{G}$ the sets of effects, F, and preconditions, G, respectively. Thus, we have $2^{|\mathcal{P}|} \cdot 2|\mathcal{P}| \cdot |\mathcal{A}|$ new propositional variables (in Section 5 we decrease this number).

Define φ_R, the logical theory that represents R, by $\varphi^0_R = \{a^F_G \in L \mid \forall \langle s, a, s' \rangle \in R,\ s \models G \Rightarrow s' \models F\}$ and $\varphi_R = \varphi^0_R \cup \{\neg a^F_G \mid a^F_G \in L \setminus \varphi^0_R\}$.

Some intuitive properties hold for this representation. φ_R is a complete theory for every R (a sentence or its negation is always implied from φ_R). Also, $\varphi_R \models \neg a_G^F \Leftrightarrow a_G^{\neg F}$, for every $F \in \mathcal{F}, G \in \mathcal{G}$.

Thus, for every transition belief state ρ we can define a theory in $\mathcal{L}(L \cup \mathcal{P})$ that corresponds to it: $\varphi_\rho = \bigvee_{\langle s,R\rangle \in \rho}(s \wedge \varphi_R)$. Similarly, for every theory φ in $\mathcal{L}(L \cup \mathcal{P})$ we define a transition belief state $\rho_\varphi = \{\langle s, R\rangle \mid s \in \mathcal{S},\ s \wedge \varphi_R \models \varphi\}$, i.e., all the state-transition pairs that satisfy φ. We say that theory φ is a *transition belief formula*, if $\varphi_{\rho_\varphi} \equiv \varphi$ (note: $\rho_{\varphi_\rho} = \rho$ always holds).

For a deterministic (possibly conditional) action, a, define the effect model of a for time t to be

$$T_{\mathsf{eff}}(a,t) = \bigwedge_{l \in \mathcal{F}, G \in \mathcal{G}}((a_t \wedge a_G^l \wedge G_t) \Rightarrow l_{t+1}) \ \wedge \ \bigwedge_{l \in \mathcal{F}}(l_{t+1} \wedge a_t \Rightarrow (\bigvee_{G \in \mathcal{G}}(a_G^l \wedge G_t))) \quad (1)$$

where a_t asserts that action a occurred at time t, and we use the convention that φ_t is the result of adding subscript t to every fluent symbol of φ. The first part of (1) says that if a executes at time t, and it causes l when G holds, and G holds at time t, then l holds at time $t+1$. The second part says that if l holds after a's execution, then it must be that a_G^l holds, with G corresponding to the current state. These two parts correspond to *effect axioms* and *explanation closure axioms* used in Situation Calculus.

Now, we are ready to describe our zeroth-level algorithm ($SLAF_0$) for SLAF of a transition belief formula. Denote by $Cn^V(\varphi)$ the set of consequences of φ that are in vocabulary V, and let $L_{t+1} = \mathcal{P}_{t+1} \cup L$ the vocabulary that includes only fluents of time $t+1$ and effect propositions from L. At time t we apply progression for the given action a and current transition belief formula, φ_t, and then apply filtering with the current observations:

1. $SLAF_0[a](\varphi_t) = Cn^{L_{t+1}}(\varphi_t \wedge a_t \wedge T_{\mathsf{eff}}(a,t))$
2. $SLAF_0[o](\varphi_t) = \varphi_t \wedge o_t$

We can implement $Cn^V(\varphi)$ using consequence finding algorithms, such as resolution and some of its variants (e.g., [Simon and del Val, 2001]). The following theorem shows that this formula-SLAF algorithm is correct and exact.

Theorem 3.1 *For φ transition belief formula, a action,*

$$SLAF[a](\{\langle s,R\rangle \in \mathfrak{S} \mid \langle s,R\rangle \textit{ satisfies } \varphi\}) = \{\langle s,R\rangle \in \mathfrak{S} \mid \langle s,R\rangle \textit{ satisfies } SLAF_0[a](\varphi)\}$$

Our zeroth-level algorithm may enable more compact representation, but it does not guarantee it, nor does it guarantee tractable computation. In fact, no algorithm can maintain compact representation or tractable computation in general. SLAF is NP-hard because Logical Filtering is NP-hard [Eiter and Gottlob, 1992] even for deterministic actions. Also, any representation of transition belief states that uses $poly(|\mathcal{P}|)$ propositions grows exponentially (in the number of time steps and $|\mathcal{P}|$) for some starting transition belief states and action sequences.

4 Efficient Model Update

Learning world models is easier when SLAF distributes over logical connectives. The computation becomes tractable, with the bottleneck being the time to compute SLAF for each part separately. In this section we examine when such distribution is possible, and present a linear-time algorithm that gives an exact solution in those cases (and a weaker transition belief formula otherwise).

Distribution properties that always hold for SLAF follow from set theoretical considerations and Theorem 3.1:

Corollary 4.1 *For φ, ψ transition belief formulae, a action,*

$$SLAF[a](\varphi \vee \psi) \equiv SLAF[a](\varphi) \vee SLAF[a](\psi)$$
$$\models SLAF[a](\varphi \wedge \psi) \Rightarrow SLAF[a](\varphi) \wedge SLAF[a](\psi)$$

Stronger distribution properties hold for SLAF whenever they hold for Logical Filtering.

Theorem 4.2 *Let ρ_1, ρ_2 be transition belief states. $SLAF[a](\rho_1 \cap \rho_2) = SLAF[a](\rho_1 \cap \rho_2)$ iff for every R $Filter[a](\rho_1^R \cap \rho_2^R) = Filter[a](\rho_1^R) \cap Filter[a](\rho_2^R)$.*

We conclude the following corollary from Theorems 3.1, 4.2 and theorems in [Amir and Russell, 2003].

Corollary 4.3 *For φ, ψ transition belief formulae, a action, $SLAF[a](\varphi \wedge \psi) \equiv SLAF[a](\varphi) \wedge SLAF[a](\psi)$ if for every relation R in ρ_φ,ρ_ψ one of the following holds:*

1. *a in R maps states 1:1*
2. *a has no conditional effects (and we know if it fails), and $\varphi \wedge \psi$ includes all its prime implicates.*
3. *The state is known for R: for at most one s, $\langle s, R\rangle \in \rho_\varphi \cup \rho_\psi$.*

Figure 2 presents Procedure Factored-SLAF, which computes SLAF exactly when the conditions of Corollary 4.3 hold. Consequently, Factored-SLAF returns an exact solution whenever our actions are known to be 1:1. If our actions have no conditional effects and their success/failure is observed, then a modified Factored-SLAF can solve this problem exactly too (see Section 5).

If we pre-compute (and cache) the $2n$ possible responses of Literal-SLAF, then every time step t in this procedure requires linear time in the representation size of φ_t, our transition belief formula. This is a significant improvement over the (super exponential) time taken by a straightforward algorithm, and over the (potentially exponential) time taken by general-purpose consequence finding used in our zeroth-level SLAF procedure above.

Theorem 4.4 *Step-SLAF(a, o, φ) returns a formula φ' such that $SLAF[a,o](\varphi) \models \varphi'$. If every run of Literal-SLAF takes time c, then Step-SLAF takes time $O(|\varphi|c)$. Finally, if we assume one of the assumptions of Corollary 4.3, then $\varphi' \equiv SLAF[a,o](\varphi)$.*

We can give a closed-form solution for the SLAF of a belief-state formula (a transition belief formulae that has no effect propositions a_G^F). This makes procedure Literal-SLAF tractable, and also allows us to examine the structure of belief state formulae in more detail.

PROCEDURE Factored-SLAF($\langle a_i, o_i \rangle_{0<i\leq t}$,$\varphi$)
$\forall i$, a_i action, o_i observation, φ transition belief formula.
1. For i from 1 to t do,
   (a) Set $\varphi \leftarrow$ Step-SLAF(o_i,a_i,φ).
   (b) Eliminate subsumed clauses in φ.
2. Return φ.

PROCEDURE Step-SLAF(o,a,φ)
o an observation formula in $\mathcal{L}(\mathcal{P})$, a an action, φ a transition belief formula.
1. If φ is a literal, then return $o \wedge$Literal-SLAF(a,φ).
2. If $\varphi = \varphi_1 \wedge \varphi_2$, return Step-SLAF($o$,$a$,$\varphi_1$)$\wedge$Step-SLAF($o$,$a$,$\varphi_2$).
3. If $\varphi = \varphi_1 \vee \varphi_2$, return Step-SLAF($o$,$a$,$\varphi_1$)$\vee$Step-SLAF($o$,$a$,$\varphi_2$).

PROCEDURE Literal-SLAF(a,φ)
a an action, φ a proposition in L_t or its negation.
1. Return $Cn^{L_{t+1}}(\varphi_t \wedge a_t \wedge T_{\mathsf{eff}}(a,t))$.

Figure 2: SLAF using distribution over $\wedge, \vee$

Theorem 4.5 *For belief-state formula $\varphi \in \mathcal{L}(\mathcal{P})$, time t, action a, $C_a = \bigwedge_{G\in\mathcal{G}, l\in\mathcal{F}}(a_G^l \vee a_G^{\neg l})$, and $G_1, ..., G_m \in \mathcal{G}$ all the terms in $\mathcal{G}$ such that $G_i \models \varphi$,*

$$SLAF[a](\varphi_t) \equiv \bigwedge_{l_1,...,l_m\in\mathcal{F}} \bigvee_{i=1}^{m} (l_i^{t+1} \vee a_{G_i}^{\neg l_i}) \wedge C_a$$

PROOF SKETCH We follow the characterization offered by Theorem 3.1 and Formula (1). We take $\varphi_t \wedge a_t \wedge T_{\mathsf{eff}}(a,t)$ and resolve out the literals of time t. This resolution is guaranteed to generate a set of consequences that is equivalent to $Cn^{L_{t+1}}(\varphi_t \wedge a_t \wedge T_{\mathsf{eff}}(a,t))$.

Assuming $\varphi_t \wedge a_t$, $T_{\mathsf{eff}}(a,t)$ is logically equivalent to $T_{\mathsf{eff}}(a,t)|_\varphi = \bigwedge_{l\in\mathcal{F},G\in\mathcal{G},G\models\varphi}((a_G^l \wedge G_t) \Rightarrow l_{t+1}) \wedge \bigwedge_{l\in\mathcal{F},G\in\mathcal{G},G\models\varphi}(l_{t+1} \Rightarrow (G_t \Rightarrow a_G^l))$. This follows from two observations. First, notice that φ_t implies that for any $G \in \mathcal{G}$ such that $G \not\models \varphi$ we get $G_t \Rightarrow a_G^l$ and $(a_G^l \wedge G_t) \Rightarrow l_{t+1}$ (the antecedent does not hold, so the formula is true). Second, notice that, in the second part of the original $T_{\mathsf{eff}}(a,t)$, $(\bigvee_{G\in\mathcal{G},G\models\varphi}(a_G^l \wedge G_t))$ is equivalent (assuming φ) to $(\bigwedge_{G\in\mathcal{G},G\models\varphi}(G_t \Rightarrow a_G^l))$.

Now, resolving out the literals of time t from $\varphi_t \wedge a_t \wedge T_{\mathsf{eff}}(a,t)|_\varphi$ should consider all the resolutions of clauses (G_t is a term) of the form $a_G^l \wedge G_t \Rightarrow l_{t+1}$ and all the clauses of the form $l_{t+1} \Rightarrow (G_t \Rightarrow a_G^l)$ with each other. This yields the equivalent to

$$\bigwedge_{\substack{G_1,...,G_m\in\mathcal{G} \\ \varphi\models\bigvee_{i\leq m}G_i \\ l_1,...,l_m\in\mathcal{F}}} [\bigvee_{i=1}^{m} l_i^{t+1} \vee \bigvee_{i=1}^{m} (a_{G_i}^{\neg l_i} \wedge \neg a_{G_i}^{l_i})]$$

because to eliminate all the literals of time t we have to resolve together sets of clauses with matching G_i's such that $\varphi \models \bigvee_{i\leq n} G_i$. The formula above encodes all the resulting clauses for a chosen set of $G_1, ..., G_m$ and a chosen set of literals $l_1, ..., l_m$. The reason for including $(a_{G_i}^{\neg l_i} \wedge \neg a_{G_i}^{l_i})$ is that we can always choose a clause with G_i, l_i of a specific type (either one that includes $\neg a_G^l$ or one that produces $a_G^{\neg l}$.

Finally, we get the formula in our theorem because $a_G^F \equiv \neg a_G^{\neg F}$ (G characterizes exactly one state in $\mathcal{S}$), and the fact that there is one set of $G_1, ..., G_m$ that is stronger than all the rest (it entails all the rest) because $G_1, ..., G_m$ are complete terms. This set is the complete fluent assignments G_i that satisfy φ. ■

A consequence of Theorem 4.5 is that we can implement Procedure Literal-SLAF using the equivalence

$$SLAF[a](l) \equiv \left\{ \begin{array}{ll} \text{Theorem 4.5} & L(l) \subseteq \mathcal{P} \\ l \wedge SLAF[a](\mathit{TRUE}) & \text{otherwise} \end{array} \right\}$$

Notice that the computation in Theorem 4.5 for $\varphi = l$ a literal is simple because $G_1, ..., G_m$ are all the complete terms in $\mathcal{L}(\mathcal{P})$ that include l.

5 Compact Model Representation

In Theorem 4.5, m could be as high as $2^{|\mathcal{P}|}$, the number of complete terms in $\mathcal{G}$. Consequently, clauses may have exponential length (in $n = |\mathcal{P}|$) and there may be a super-exponential number of clauses in this result.

In this section we restrict our attention to STRIPS actions [Fikes *et al.*, 1972], and provide an overall polynomial bound on the growth of our representation, its size after many steps, and the time taken to compute the resulting model. STRIPS is a class of actions that has been at the focus of interest for the area of AI planning and execution for many years. It includes deterministic, unconditional (but sometimes not executable) actions.

5.1 Actions of Limited Effect

In many domains we can assume that every action a affects at most k fluents, for some small $k > 0$. It is also common to assume that our actions are STRIPS, and that they may fail without us knowing, leaving the world unchanged. Those assumptions together allow us to progress SLAF with a limited (polynomial factor) growth in the formula size.

We use a language that is similar to the one in Section 3, but which uses only action propositions a_G^l with G being a fluent term of size k (instead of a fluent term of size n in $\mathcal{G}$). Semantically, $a_{l_1\wedge...\wedge l_k}^l \equiv \bigwedge_{l_{k+1},...,l_n} a_{l_1\wedge...\wedge l_n}^l$.

Theorem 5.1 *Let $\varphi \in \mathcal{L}(\mathcal{P})$ be a belief-state formula, t time, and a a STRIPS action with $\leq k$ fluents affected or in the precondition term. Let $\mathcal{G}^k$ be the set of all terms of k fluents in $\mathcal{L}(\mathcal{P})$ that are consistent with φ. Then,*

$$SLAF[a](\varphi_t) \equiv \bigwedge_{\substack{G_1,...,G_k\in\mathcal{G}^k \\ G_1\wedge...\wedge G_k\models\varphi \\ l_1,...,l_k\in\mathcal{F}}} \bigvee_{i=1}^{k} (l_i^{t+1} \vee a_{G_i}^{\neg l_i}) \wedge C_a$$

The proof (omitted) uses two insights. First, if a has only one case in which change occurs, then every clause in Theorem 4.5 is subsumed by a clause that is entailed by $SLAF[a](\varphi_t)$, has at most one $a^{l_i}_{G_i}$ per literal l_i (i.e., $l_i \neq l_j$ for $i \neq j$) and G_i is a fluent term (has no disjunctions). Second, every a^l_G with G term is equivalent to a formula on $a^l_{G_i}$ with G_i terms of length k, if a affects only k fluents.

Thus, we can encode all of the clauses in the conjunction using a subset of the (extended) action effect propositions, a^l_G, with G being a term of size k. There are $O(n^k)$ such terms, and $O(n^{k+1})$ such propositions. Every clause is of length $2k$, with the identity of the clause determined by the first half (the set of action effect propositions). Consequently, $SLAF[a](\varphi_t)$ takes $O(n^{k^2+k} \cdot k^2)$ space to represent using $O(n^{k^2+k})$ clauses of length $\leq 2k$.

5.2 Always-Executable Actions

Many times we can assume that our actions are always executable. For example, our sequence of actions does not include action failures when this sequence is chosen by a knowledgeable agent (e.g., a human) that has better observations than we do.

We use a language that is similar to the one in Section 3, but which uses only action propositions a^l (l always holds after executing a) and $a^{l\circ}$ (l is not affected by a). Semantically, $a^l \equiv \bigwedge_{l_1,...,l_n} a^l_{l_1 \wedge ... \wedge l_n}$, and $a^{l_i\circ} \equiv \bigwedge_{l_1,...,l_{i-1},l_{i+1},...,l_n} a^{l_i}_{l \wedge l_1 \wedge ... \wedge l_n}$.

We assume that transition-belief formula φ is *fluent-factored*, i.e., it is the conjunction of φ_f such that each φ_f concerns only one fluent, f, and actions' effects on it. Also, for every f, $\varphi_f \equiv (\neg f \vee \textit{expl}_f) \wedge (f \vee \textit{expl}_{\neg f}) \wedge A_f$, with $\textit{expl}_f, \textit{expl}_{\neg f}, A_f$ formulae over action propositions a^f, $a^{\neg f}$, and $a^{f\circ}$ (possibly multiple different actions). We call this format *fluent-factored*. The intuition here is that $\textit{expl}_f$ and $\textit{expl}_{\neg f}$ are all the possible explanations for f being true and false, respectively. Also, A_f holds knowledge about actions' effects on f, knowledge that does not depend on f's current value. For example, if we know nothing about actions that affect f (e.g., when we start our exploration), then $\varphi_f = (\neg f \vee TRUE) \wedge (f \vee TRUE) \wedge TRUE$.

With this representation, $SLAF[a](\varphi_f)$ is

$$(\neg f \vee (a^f \vee (a^{f\circ} \wedge \textit{expl}_f))) \wedge (f \vee (a^{\neg f} \vee (a^{f\circ} \wedge \textit{expl}_{\neg f}))) \wedge A_f$$

A similar formula holds for observations.

This contributes to a much simpler algorithm, AE-STRIPS-SLAF, that we present in Figure 3. It computes SLAF of a fluent-factored transition belief state in this form with a sequence of actions and observations.

Theorem 5.2 *Let φ be a fluent-factored transition belief formula, and assume $a_1, ..., a_T$ are always-executable STRIPS actions, and $o_1, ..., o_T$ are observation terms. Then, Procedure AE-STRIPS-SLAF($\langle a_i, o_i \rangle_{0<i\leq t}$,$\varphi$) returns $\varphi^T \equiv SLAF[\langle a_i, o_i \rangle_{0<i\leq t}](\varphi_0)$ in time $O(T \cdot |\mathcal{P}|)$, and $|\varphi^T| \leq O(T \cdot |\mathcal{P}|)$.*

PROCEDURE AE-STRIPS-SLAF($\langle a_i, o_i \rangle_{0<i\leq T}$,$\varphi$)
a_i actions, o_i observations, $\varphi = \bigwedge_{f\in\mathcal{P}} \varphi_f$ fluent-factored with $\varphi_f = (\neg f \vee \textit{expl}_f) \wedge (f \vee \textit{expl}_{\neg f}) \wedge A_f$.

1. For every $f \in \mathcal{P}$ do: For i from 1 to T do,
   (a) Set $\varphi_f \leftarrow (\neg f \vee (a^f \vee (a^{f\circ} \wedge \textit{expl}_f))) \wedge (f \vee (a^{\neg f} \vee (a^{f\circ} \wedge \textit{expl}_{\neg f}))) \wedge A_f$.
   (b) If $o_i \models f$ (we observed f), then set $\varphi_f \leftarrow (\neg f \vee TRUE) \wedge (f \vee FALSE) \wedge A_f \wedge \textit{expl}_f$.
   (c) If $o_i \models \neg f$ (we observed $\neg f$), then set $\varphi_f \leftarrow (\neg f \vee FALSE) \wedge (f \vee TRUE) \wedge A_f \wedge \textit{expl}_{\neg f}$.
2. Eliminate subsumed clauses in $\varphi = \bigwedge_{f\in\mathcal{P}} \varphi_f$.
3. Return $\varphi_T = \varphi$.

Figure 3: SLAF with always-executable STRIPS.

The reason for the space bound on the end formula $|\varphi^T|$ is that at every time step we increase the size of the formula by adding to it at most 4 new elements per fluent.

We can further improve the implementation to take time $O(T \cdot |o|)$, for $|o|$ the largest number of fluents observed at once. Also, the bound on the space taken by φ_T can be improved in two ways. First, $O(|\mathcal{P}| \cdot 2^{|\mathcal{A}| \log |\mathcal{A}|})$, for $\mathcal{A}$ our set of actions, is a bound on the CNF representation of φ^T because every part φ^T_f of it is similar to $(f \Rightarrow (a^f_1 \vee (a^{f\circ}_1 \wedge (a^f_2 \vee (a^{f\circ}_2 \wedge ...)))))$. Thus, if $|\mathcal{A}|$ is small, then this bound is manageable. Second, if every fluent f is observed at least once every k actions (or an action which affects f occurs, and we know its effect), then the CNF representation of φ_T takes space $O(|\mathcal{P}| \cdot |\mathcal{A}|^k)$. Again, if k is small, this bound is useful.

5.3 Action Preconditions

Unfortunately, a simple procedure such as AE-STRIPS-SLAF (Figure 3) does not provide an exact solution in general. The reason is that action failure makes fluents co-dependent (each of them could have caused this failure). For instance, consider Figure 1, and assume that we add the fluent OK which indicates that the last action succeeds. If we know only $\neg on$ (the light is off), then $\textit{SLAF}[\textit{sw-on}](\neg on) \wedge \neg OK$ (i.e., *sw-on* failed, e.g., because we are in the West room) implies $sw \vee E \vee a^{\neg OK}_{\neg E \wedge \neg sw \wedge \neg lit} \vee a^{\neg sw}_{\neg E \wedge sw \wedge \neg lit} \vee a^{\neg E}_{E \wedge \neg sw \wedge \neg lit} \vee a^{\neg OK}_{E \wedge sw \wedge \neg lit}$, by Theorem 4.5.

Instead, we compute the effect for every literal using Theorem 5.1, (assuming exactly k fluents are changed by action a) and then approximate by dropping all clauses with more than one fluent (other than OK). Thus, we can augment step 1(a) in AE-STRIPS-SLAF with a condition that a succeeded ($o_i \models OK$), and add that if a fails, then we use the following formula instead: Set $\varphi_f \leftarrow (\neg f \vee (\textit{expl}_f \wedge \textit{expl}_{\neg OK,f})) \wedge (f \vee (\textit{expl}_{\neg f} \wedge \textit{expl}_{\neg OK,\neg f})) \wedge A_f$,

for $\mathit{expl}_{\neg OK,l} = \bigwedge_{\substack{G_1,G_2 \in \mathcal{G}^k \\ G_1 \wedge G_2 \models l}} (a^l_{G_1} \vee a^{\neg OK}_{G_2})$.

The size of the resulting formula is larger by an additive $O(|\mathit{expl}_{\neg OK,l}|) = O(n^{2k})$ per fluent. Thus, the result after T steps is a formula φ_T with $|\varphi_T| \leq O(T \cdot |\mathcal{P}|^{2k})$, which is polynomial and feasible for small k's.

5.4 Using the Model and Adding Bias

Our algorithms compute a solution to SLAF as a logical formula. We can use a SAT solver to answer queries over this formula, such as checking if it entails a^f, for action a and fluent f. The number of variables in the result formula is *always independent of* T, and is linear in $|\mathcal{P}|$ for some of our algorithms, so we can use current SAT solvers to treat domains of 1000 features and more.

Many times it is also desirable to prefer some models over others. We can represent such bias using a preference model (e.g., [McCarthy, 1986]) or a probabilistic prior over transition relations. We add this bias on top of our SLAF result, φ_T, and get an exact solution if we can run inference on the combined model. Preferential bias is well studied and fits easily with logical formula φ_T (e.g., we can use implementations of Circumscription for inference with such bias).

Also, algorithms for inference with probabilistic bias and logical sentences are now emerging and can be used here too [Hajishirzi and Amir, 2005]. We can use these algorithms to apply probabilistic bias to the resulting logical formula. For example, given a probabilistic graphical model (e.g., Bayesian Networks) and a set of propositional logical sentences, we can consider the logical sentences as observations. With this approach, a logical sentence φ gives rise to a characteristic function $\delta_\varphi(\overrightarrow{x})$ which is 1 when $\overrightarrow{x}$ satisfies φ and 0 otherwise. For a conjunction of clauses we get a set of such functions (one per clause). Thus, inference in the combined probabilistic-logical system is a probabilistic inference (such as variable elimination (e.g., [Dechter, 1999])) in which there are additional potential functions.

6 Experimental Evaluation

We tested our AE-STRIPS-SLAF algorithm in partially observable blocks-world domains. We generated random action-observation sequences of 4000 steps from a description of the domain in PDDL. Our SLAF algorithm receives no domain information besides the sequence of actions and observations that is given by the generator.

We ran the algorithm with blocks-world domains of increasing sizes (31 to 157 features). We collected the time and space taken for each time step in the execution, starting with zero knowledge. The results are shown in Figure 4. Each of the graphs belongs to a different domain size. They show that the time per step approaches a (very small, order of < 9 milliseconds) limit as the execution proceeds. The time that is taken to perform SLAF of different domains grows linearly with domain size.

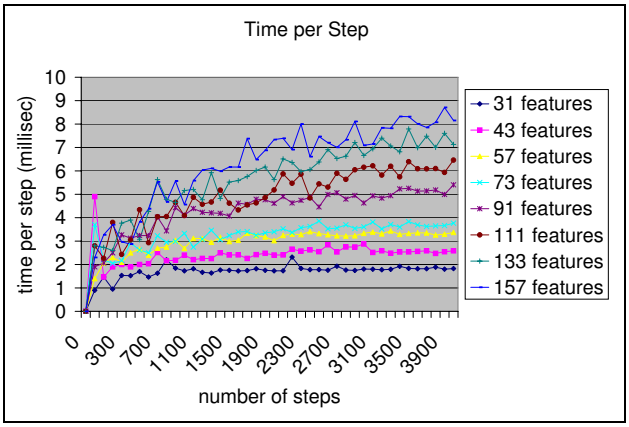

Figure 4: AE-STRIPS-SLAF in the blocks world.

Also, [Hlubocky and Amir, 2005] has included a modified version of this algorithm in their architecture and tested it on a suite of adventure-game-like virtual environments that are generated at random. These include arbitrary numbers of places, objects of various kinds, and configurations and settings of those. There, an agent's task is to exit a house, starting with no knowledge about the state space, available actions and their effects, or characteristics of objects. Their experiments show that the agent learns the effects of its actions very efficiently. This agent makes decisions using the learned knowledge, and inference with the resulting representation is fast (a fraction of a second per SAT problem in domains including more than 30 object, modes, and locations).

7 Comparison with Related Work

HMMs [Boyen and Koller, 1999; Boyen *et al.*, 1999; Murphy, 2002; Ghahramani, 2001] can be used to estimate a stochastic transition model from observations. Unfortunately, HMMs require an explicit representation of the state space, and our smallest domain (31 features) requires a transition matrix of $(2^{31})^2$ entries. This prevents initializing HMMs procedures on any current computer.

Structure learning approaches in Dynamic Bayes Nets (DBNs and also fHMMs) use EM and additional approximations (e.g., using factoring, variation, or sampling), and are more tractable. However, they are still limited to small domains (e.g., 10 features [Ghahramani and Jordan, 1997; Boyen *et al.*, 1999]), and also have unbounded errors in discrete deterministic domains, so are not usable in our settings.

Reinforcement Learning (RL) approaches compute a mapping between world states and preferred actions. They are highly intractable in partially observable domains [Kaelbling *et al.*, 1998], and approximation (e.g., [Kearns *et al.*, 2000] is practical only for small domains (e.g., 10-20 features) with small horizon time T.

In contrast to HMMs, DBNs, and RL, our algorithms are exact and tractable in large domains (> 300 features). We take advantages of properties common to discrete domains, such as determinism, limited effects of actions, and observed failure.

Previous work on learning action models assumes *fully observable* deterministic domains. These learn parametrized STRIPS actions using, e.g., version spaces [Gil, 1994; Wang, 1995], general classifiers [Oates and Cohen, 1996], or hill-climbing ILP [Benson, 1995]. Recently, [Pasula *et al.*, 2004] gave an algorithm that learns stochastic actions with no conditional effects. [Schmill *et al.*, 2000] approximates partial observability by assuming that the world is fully observable. We can apply these to partially observable learning problems (sometimes) by using the space of belief states instead of world states, but this increases the problem size to exponentially, so this is not practical for our problem.

8 Conclusions

We presented a framework for learning the effects and preconditions of deterministic actions, and showed that in several common situations this can be done exactly in time that is polynomial (sometime linear) in the number of time steps and features. We can add bias and compute an exact solution for large domains (hundreds of features), in many cases.

The results that we presented are promising for many applications, including reinforcement learning, agents in virtual domains, and HMMs. Already, this work is being applied to autonomous agents in adventure games, and exploration is guided by the transition belief state that we compute and information gain criteria. In the future we plan to extend these results to stochastic domains, and domains with continuous features.

9 Acknowledgements

I wish to thank Megan Nance for providing the code and samples for the generator. I also wish to acknowledge support from DAF Air Force Research Laboratory Award FA8750-04-2-0222 (DARPA REAL program).

References

[Amir and Russell, 2003] Eyal Amir and Stuart Russell. Logical filtering. In *IJCAI '03*, pages 75–82. MK, 2003.

[Benson, 1995] Scott Benson. Inductive learning of reactive action models. In *Proc. ICML-94*, 1995.

[Boyen and Koller, 1999] Xavier Boyen and Daphne Koller. Approximate learning of dynamic models. In Michael S. Kearns, Sara A. Solla, and David A. Kohn, editors, *NIPS 1998*, pages 396–402. Cambridge: MIT Press, 1999. Available at http://www.cs.stanford.edu/ xb/nips98/.

[Boyen *et al.*, 1999] Xavier Boyen, Nir Friedman, and Daphne Koller. Discovering the hidden structure of complex dynamic systems. In *Proc. UAI '99*, pages 91–100. MK, 1999.

[Dechter, 1999] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

[Eiter and Gottlob, 1992] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *AIJ*, 57(2-3):227–270, 1992.

[Fikes *et al.*, 1972] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *AIJ*, 3, 1972.

[Ghahramani and Jordan, 1997] Z. Ghahramani and M. I. Jordan. Factorial Hidden Markov Models. *Machine Learning*, 29:245–275, 1997.

[Ghahramani, 2001] Zoubin Ghahramani. An introduction to Hidden Markov Models and Bayesian networks. *Intl J. of Pattern Recog. and AI*, 15(1):9–42, 2001.

[Gil, 1994] Yolanda Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proc. ICML-94*, pages 10–13, 1994.

[Hajishirzi and Amir, 2005] Hannaneh Hajishirzi and Eyal Amir. Tractable exact inference in some dynamic Bayesian networks. In *submitted for publication*, 2005.

[Hlubocky and Amir, 2005] Brian Hlubocky and Eyal Amir. Logic-based approach to decision making in expanding worlds. In *submitted for publication*, 2005.

[Kaelbling *et al.*, 1998] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *AIJ*, 101:99–134, 1998.

[Kearns *et al.*, 2000] M. Kearns, Y. Mansour, and A. Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Proc. NIPS'99*, pages 1001–1007, 2000.

[Littman, 1996] M. L. Littman. *Algorithms for sequential decision making*. PhD thesis, Brown U., 1996.

[McCarthy, 1986] John McCarthy. Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[Murphy, 2002] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, 2002.

[Oates and Cohen, 1996] T. Oates and P. R. Cohen. Searching for planning operators with context-dependent and probabilistic effects. In *Proc. AAAI '96*, 1996.

[Pasula *et al.*, 2004] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning probabilistic relational planning rules. In *Proc. ICAPS'04*, 2004.

[Schmill *et al.*, 2000] M. D. Schmill, T. Oates, and P. R. Cohen. Learning planning operators in real-world, partially observable environments. In *Proc. AIPS'00*, 2000.

[Simon and del Val, 2001] Laurent Simon and Alvaro del Val. Efficient consequence-finding. In *IJCAI '01*, 2001.

[Wang, 1995] Xuemei Wang. Learning by observation and practice: an incremental approach for planning operator acquisition. In *Proc. ICML-95*, pages 549–557. MK, 1995.