Efficient Learning of Action Models for Planning

Neville Mehta and **Prasad Tadepalli** and **Alan Fern**
School of Electrical Engineering and Computer Science
Oregon State University, Corvallis, OR 97331, USA.
{mehtane,tadepall,afern}@eecs.oregonstate.edu

Abstract

We consider the problem of learning action models for planning in two frameworks and present general sufficient conditions for efficient learning. In the *mistake-bounded planning* framework, the learner has access to a sound and complete planner for the given action model language, a simulator, and a planning problem generator. In the *planned exploration* framework, the learner has access to a planner and a simulator, but actively generates problems to help refine its model. We identify sufficient conditions for learning in both the frameworks. We also show that a concrete hypothesis space that consists of sets of rules with at most k variables is efficiently learnable in both frameworks.

Introduction

Planning research typically assumes that the planning system has access to complete and correct models of the actions. However, that raises the obvious question: where do the models come from? In this paper, we formulate and analyze the question of learning action models suitable for planning. Since the agents might need to plan even before complete and correct models are learned, model learning, planning, and plan execution must be interleaved in an autonomous agent.

We focus our attention on learning deterministic action models for planning for goal achievement. The deterministic planning setting will let us explore strong success criteria, namely, a worst-case polynomial bound on mistakes or a polynomial number of planning attempts before convergence. It has been shown that deterministic STRIPS actions with a constant number of preconditions can be learned from raw experience with at most a polynomial number of plan prediction mistakes (Walsh and Littman 2008). In spite of the above positive results, compact action models in fully observable, deterministic action models are not always learnable. For example, action models represented as arbitrary Boolean functions are not learnable under standard cryptographic assumptions such as the hardness of factoring. Further, we require that the learner to learn only from self-generated plans, which further limits what can be learned. Instead of selecting an action at every step to approximately optimize the long-term reward as in the PAC-MDP algorithms, our learner is expected to solve problems by generating plans and executing them. We define two distinct frameworks for learning action models for planning, and characterize sufficient conditions for success in these frameworks.

Learning action models for planning is different from learning an arbitrary function from states and actions to next states because the learner has some control over the actions it executes, giving it the freedom to ignore modeling the effects of some actions in certain contexts. For example, most people who drive do not ever learn a complete model of the dynamics of their cars; while they might accurately know the stopping distance or turning radius, they could be oblivious to many aspects that an expert auto-mechanic is comfortable with. To capture this intuition, we introduce the concept of an *adequate* model, that is, a model that is sufficiently complete and correct for planning for a given class of goals. For example, one need not know a complete map of a city to navigate effectively. In most cases, it suffices to learn one route for the places one needs to go to. In other words, any spanning tree of the graph of the city over the goals and starting points of interest would be an adequate model.

In the *mistake-bounded planning* (MBP) framework, the goal is to keep solving user-generated planning problems while learning action models and guarantee at most a polynomial number of mistakes or unsuccessful plans. While polynomial number of mistakes is not always reasonable, e.g., when flying real helicopters to learn their dynamics, the goal here is to characterize the minimal structure of problems that lends itself to autonomous learning when mistakes are relatively cheap. We assume that in addition to the problem generator, the learner has access to a sound and complete planner and a simulator (or the real world). We give general sufficient conditions for learning an adequate model with a polynomial mistake bound.

In the spirit of self-directed learning, we also introduce the *planned exploration* (PLEX) framework, where the learner needs to generate its own problems to solve to refine its action model. This requirement translates to an experiment-design problem, where the learner needs to design problems in a goal language which help it disambiguate the action models. We also identify a set of general sufficient conditions for efficient learning in this framework.

Our sufficient conditions are based on learning schemas that maintain an optimistic action model which includes all transitions of an adequate model. Given such an optimistic model, the correct plan for any problem can always be generated by a sound and complete planner. However, many incorrect plans may also be generated. The idea behind our approach is to simulate the plans generated by the planner, collect examples of all observed actions, and use them to refine the action models. In doing so, we fully depend on determinism. In particular, action models are refined by ruling out all outcomes other than those that actually happened for a given state-action pair. In the PLEX framework, problems are generated internally by the learner, which is driven by the purpose of disambiguating conflicting predictions in the models.

We consider a specific language, k-SAP (sets of action productions of at most k variables), and show that it is learnable in polynomial time in both the MBP and the PLEX frameworks for an appropriate goal language.

Formal Preliminaries

A factored planning domain $\mathcal{P}$ is a tuple (V, D, A, T), where $V = \{v_1, \ldots, v_n\}$ is the set of variables, D is the domain of the variables in V, and A is the set of actions. $S = D^n$ represents the state space, and $T \subset S \times A \times S$ is the transition relation where $(s, a, s') \in T$ signifies that taking action a in state s results in state s'. The domain parameters, $n, |D|$, and $|A|$, characterize the size of $\mathcal{P}$ and are implicit in all claims of complexity in the rest of this paper.

Action Models and Hypothesis Spaces

We only consider learning deterministic action models. Hence, the transition relation is in fact a function, although the learner's hypothesis space includes nondeterministic models.

Definition 1. *An* action model *is a relation $M \subseteq S \times A \times S$.*

This work emphasizes model learning via interaction with a simulator. The set of positive examples of the transition function observed via experience is $Z \subseteq T$. Because T is deterministic, every positive example (s, a, s') implicitly entails several negative examples $\{(s, a, s'') : s'' \neq s'\}$; we let Z^- denote the set of all negative examples given Z.

Definition 2. *A model M is* weakly consistent *with a set of examples Z if $M \cap Z^- = \varnothing$. It is* strongly consistent *with Z if, in addition, $M \supseteq Z$.*

We consider compact representations of action models in this paper.

Definition 3. *A* hypothesis *is a representation of an action model. The* hypothesis space *$\mathcal{H}$ of action models is the language of all such hypotheses considered by the learner. Given an example set Z, the* version space *of action models is the subset of all hypotheses in $\mathcal{H}$ that are weakly consistent with Z and is denoted as $\mathcal{M}(Z)$.*

Note that the hypotheses in the version space are only defined to be weakly consistent. This means that our models may not include all observed positive transitions in them, although they must exclude their negative implications. This will be important to allow us to ignore some action models that are not needed for successful planning. With some abuse of notation, we use the words hypothesis and model interchangeably. We only consider finite (possibly parameterized) hypothesis spaces.

Without loss of generality, $\mathcal{H}$ can be structured as a *generalization graph* where the nodes correspond to sets of equivalent hypotheses (represent the same set of transitions) and there is a directed edge from node n_1 to node n_2 if and only if the model that corresponds to n_1 is strictly more general than (a strict superset of) the model that corresponds to n_2.

Definition 4. *The* height *of $\mathcal{H}$ is the length of the longest path from a root node to a leaf node in its generalization graph.*

Definition 5. *$\mathcal{H}$ is* well-structured *if, for any example set Z of some true model, the version space $\mathcal{M}(Z)$ has a most general hypothesis $mgh(Z)$. Further, if there exists an algorithm that can compute $mgh(Z \cup \{z\})$ from $mgh(Z)$ and a new example z in time polynomial in the size of $mgh(Z)$ and z, then we say that $\mathcal{H}$ is* efficiently well-structured.

Note that it follows from the definition that all most general hypotheses represent the same model, i.e., the set of transitions. This is also called the *optimistic model* because it includes every transition in every model in $\mathcal{M}(Z)$. If $\mathcal{H}$ is well-structured, then its generalization graph has a unique root node which corresponds to the optimistic model of $\mathcal{H}$. It turns out that well-structuredness is easier to verify if it satisfies the following property.

Definition 6. *A hypothesis space $\mathcal{H}$ is* closed under union *if $M_1, M_2 \in \mathcal{H} \implies M_1 \cup M_2 \in \mathcal{H}$.*

Lemma 1. *$\mathcal{H}$ is well-structured if $\mathcal{H}$ is finite and closed under union.*

Proof. Let $H_0 = \bigcup_{M \in \mathcal{M}(Z)} M$ represent the unique union of all models represented by hypotheses in $\mathcal{M}(Z)$. Because $\mathcal{H}$ is finite and closed under union, H_0 must be in $\mathcal{H}$. If $\exists z \in H_0 \cap Z^-$, then $z \in M \cap Z^-$ for some $M \in \mathcal{M}(Z)$. This is a contradiction since all $M \in \mathcal{M}(Z)$ are weakly consistent with Z. Consequently, H_0 is weakly consistent with Z, and is in $\mathcal{M}(Z)$. It is more general than (is a superset of) every other hypothesis in $\mathcal{M}(Z)$ because it is their union. □

Planning Components

Our action models are intended for planning, which is captured by the following definitions.

Definition 7. *A* planning problem *is a pair (s_0, g) where $s_0 \in S$ and the goal condition g is an expression chosen from a goal language $\mathcal{G}$ and represents a set of states in which it evaluates to true. A state s* satisfies *a goal g if and only if g is true in s.*

Definition 8. *Given a planning problem (s_0, g), a* plan *is a sequence of actions $a_1, \ldots, a_p$. The plan is* correct *w.r.t. (M, g) if $\exists s_1, \ldots, s_p$ such that $(s_{i-1}, a_i, s_i) \in M$ for $1 \leq i \leq p$ and the state s_p satisfies the goal g.*

Definition 9. *A* planner *for the hypothesis-goal space $(\mathcal{H}, \mathcal{G})$ is an algorithm that takes $M \in \mathcal{H}$ and $(s_0, g \in \mathcal{G})$ as inputs and outputs a plan or signals failure. It is* sound *w.r.t. $(\mathcal{H}, \mathcal{G})$ if, given any M and (s_0, g), it produces a correct plan w.r.t. (M, g) or signals failure. It is* complete *w.r.t.*

$(\mathcal{H}, \mathcal{G})$ if, given any M and (s_0, g), it produces a correct plan whenever one exists w.r.t. (M, g).

Note that we generalize the definition of soundness from its standard usage in the literature in order to apply to nondeterministic action models, where the nondeterminism is "angelic" — the planner can control the outcome of actions when multiple outcomes are possible according to its model (Marthi, Russell, and Wolfe 2007). One way to implement such a planner is to do forward search through all possible action and outcome sequences and return an action sequence if it leads to a goal under some outcome choices. Our analysis is agnostic to plan quality or plan length and applies equally well to suboptimal planners. This is motivated by the fact that optimal planning is hard for most domains, but suboptimal planning such as hierarchical planning can be quite efficient.

We now describe the concept of an adequate action model for a class of goals.

Definition 10. *Let $\mathcal{P}$ be a planning domain and $\mathcal{G}$ be a goal language. An action model M is* adequate *for $\mathcal{G}$ in $\mathcal{P}$ if $M \subseteq T$ and the existence of a correct plan w.r.t. (T, g) implies the existence of a correct plan w.r.t. (M, g). $\mathcal{H}$ is adequate for $\mathcal{G}$ if $\exists M \in \mathcal{H}$ such that M is adequate for $\mathcal{G}$.*

An adequate model may be partial or incomplete in that it may not include every possible transition in the transition function T. However, the model is sufficient to produce a correct plan w.r.t. (T, g) for every goal g in the desired class. Thus, the more limited the goal class, the more incomplete the adequate model can be. In the example of a city map, if the goal language excludes certain locations, then so can the spanning tree.

Definition 11. *A* simulator *of the domain is always situated in the current state s. It takes an action a as input, transitions to the state s' resulting from executing a in s, and returns the current state s'.*

Definition 12. *Given a goal language $\mathcal{G}$, a* problem generator *generates an arbitrary problem $(s_0, g \in \mathcal{G})$ and sets the state of the simulator to s_0.*

Mistake-Bounded Planning Framework

To introduce and prove a general theorem that characterizes the mistake-bounded planning (MBP) framework, we first define what it means to make a mistake.

Definition 13. *A* planning mistake *occurs if the planner signals failure when a correct plan exists w.r.t. the transition function T or when the plan output by the planner is not sound w.r.t. T.*

Definition 14. *Let $\mathcal{G}$ be a goal language for which $\mathcal{H}$ is an adequate hypothesis space. $\mathcal{H}$ is* learnable in the MBP framework *if there exists an algorithm $\mathcal{A}$ that interacts with a problem generator over $\mathcal{G}$, a sound and complete planner w.r.t. $(\mathcal{H}, \mathcal{G})$, and a simulator of the planning domain $\mathcal{P}$, and outputs a plan or signals failure for each planning problem while guaranteeing at most a polynomial number of planning mistakes. Further, $\mathcal{H}$ is* polynomial-time learnable *in the MBP framework if $\mathcal{A}$ always responds in time polynomial in the domain parameters and the length of the longest plan generated by the planner, assuming that a call to the planner, simulator, or problem generator takes $O(1)$ time.*

Algorithm 1 MBP LEARNING SCHEMA
Input: Hypothesis space $\mathcal{H}$, goal language $\mathcal{G}$

```
 1: $M \leftarrow$ optimistic model of $\mathcal{H}$
 2: loop
 3:     $(s, g) \leftarrow$ PROBLEMGENERATOR($\mathcal{G}$)
 4:     $plan \leftarrow$ PLANNER($M, (s, g)$)
 5:     if $plan \neq$ failure then
 6:         for $a$ in $plan$ do
 7:             $s' \leftarrow$ SIMULATOR($a$)
 8:             $M \leftarrow$ MODELLEARNER($M, (s, a, s')$)
 9:             $s \leftarrow s'$
10:         if $s$ satisfies $g$ then
11:             print $plan$
12:         else
13:             print fail
```

Along with the domain description, the learner is given a hypothesis space which is guaranteed to contain an adequate model for the goals in a goal language. Importantly, the hypothesis space need not contain the true transition function because an adequate model is good enough for planning. The goal of the learner is to determine such a model by continually maintaining an optimistic model of the version space. It does this by excluding from the optimistic model any transitions that conflict with the positive observations. However, it may not necessarily contain all positive observations in its optimistic model. Note that we cannot bound the time for the convergence of $\mathcal{A}$ because there is no limit on when the mistakes are made.

Theorem 1. *$\mathcal{H}$ is learnable in the MBP framework if it is well-structured, has polynomial height, and is adequate for the desired goal language.*

Proof. Algorithm 1 is a general schema for action model learning in the MBP framework. The current model M is initialized to the optimistic model of the hypothesis space $\mathcal{H}$; this is guaranteed to exist due to $\mathcal{H}$ being well-structured. PROBLEMGENERATOR provides a planning problem and initializes the current state of SIMULATOR. Given M and the planning problem, PLANNER always outputs a plan if one exists because $\mathcal{H}$ contains a "target" adequate model that is weakly consistent with the observations and is always retained in the version space; the optimistic model used by PLANNER is more general than any such target model. If PLANNER signals failure, then there is no plan for it; otherwise, the plan is executed through SIMULATOR and MODELLEARNER uses every transition (s, a, s') to refine the model M making sure that it does not include any transitions in $\{(s, a, s'') : s'' \neq s'\}$. The resulting model is the most general possible that excludes the illegal transitions. The plan is output if the final state satisfies the goal. Because the maximum number of model refinements is bounded by the height of $\mathcal{H}$, the number of planning mistakes is polynomial. Thus, $\mathcal{H}$ is learnable in the MBP framework. $\square$

The algorithm ensures that some adequate model always remains as a specialization of the optimistic model. As ModelLearner checks only for weak consistency, it might eliminate models from the version space that do not include the observed positive transitions. This does not hurt the algorithm as it only seeks to learn an adequate model, not an exact one.

The above result generalizes the work on learning STRIPS operator models from raw experience (without a teacher) in Walsh and Littman (2008) to arbitrary hypotheses spaces by identifying sufficiency conditions. (A hypothesis class considered later in this paper subsume propositional STRIPS by capturing conditional effects.) It also clarifies the notion of adequate models, which can be much simpler than the true transition model, and the influence of the goal language on the complexity.

Corollary 1. *A hypothesis space $\mathcal{H}$ is polynomial-time learnable in the MBP framework if it is efficiently well-structured, has polynomial height, and is adequate for the desired goal language.*

Proof. This follows from the fact that all components in Algorithm 1 other than ModelLearner are assumed to run in $O(1)$ time. □

Planned Exploration Framework

The MBP framework is appropriate when mistakes are permissible on user-given problems as long as their total number is limited. It is not appropriate in cases where no mistakes are permitted after an initial training period. We overcome this limitation in the planned exploration (PLEX) framework, where the agent seeks to learn an action model for the domain without an external problem generator. It generates planning problems for itself based on a goal language and solves for them. The key issue here is to generate a reasonably small number of planning problems such that solving them would identify a deterministic action model.

Learning a model in the PLEX framework involves knowing where and how it is deficient and then planning to reach states that are informative, which entails formulating planning problems in a goal language. This framework provides a polynomial sample convergence guarantee which is stronger than a polynomial mistake bound of the MBP framework. Without a problem generator that can change the simulator's state, it is impossible for the simulator to transition freely between strongly connected components (SCCs) of the transition graph. Hence, we make the assumption that the transition graph is a disconnected union of SCCs and require only that the agent learn the model for a single SCC that contains the initial state of the simulator.

Definition 15. *Let $\mathcal{P}$ be a planning domain whose transition graph is a union of SCCs, and let $\mathcal{H}$ be an adequate hypothesis space for the goal language $\mathcal{G}$. $(\mathcal{H}, \mathcal{G})$ is* learnable in the PLEX framework *if there exists an algorithm $\mathcal{A}$ that interacts with a sound and complete planner w.r.t. $(\mathcal{H}, \mathcal{G})$ and the simulator for $\mathcal{P}$ and outputs a model $M \in \mathcal{H}$ that is adequate within the SCC that contains the initial state s_0 of the simulator after a polynomial number of planning attempts. Further, $(\mathcal{H}, \mathcal{G})$ is* polynomial-time learnable *in the PLEX framework if $\mathcal{A}$ runs in polynomial time in the domain parameters and the length of the longest plan output by the planner, assuming that every call to the planner and the simulator take $O(1)$ time.*

A key step in planned exploration is designing appropriate planning problems. We call these *experiments* as the goal of solving these problems is to disambiguate nondeterministic action models. In particular, the agent tries to reach an *informative* state where the current model predicts two different next states for the same action.

Definition 16. *Given a model M, the set of* informative states *is $I(M) = \{s : (s, a, s'), (s, a, s'') \in M \wedge s' \neq s''\}$, where a is said to be* informative *in s.*

Definition 17. *A set of goals G is a* cover *of a set of states R if $\bigcup_{g \in G}\{s : s \text{ satisfies } g\} = R$.*

Given the goal language $\mathcal{G}$ and a model M, the problem of experiment design is to find a set of goals $G \subseteq \mathcal{G}$ such that the sets of states that satisfy the goals in G collectively cover all informative states $I(M)$. If it is possible to plan to achieve one of these goals, then either the plan passes through a state where the model is nondeterministic or it executes successfully and the agent reaches the final goal state; in either case, an informative action can be executed and and observed transition is used to refine the model. If none of the goals in G can be successfully planned for, then no informative states for that action are reachable. We formalize these intuitions below.

Definition 18. *The* width *of $(\mathcal{H}, \mathcal{G})$ is defined as*

$$\max_{M \in \mathcal{H}} \min_{G \subseteq \mathcal{G} : G \text{ is a cover of } I(M)} |G|$$

where $\min_G |G| = \infty$ if there is no $G \subseteq \mathcal{G}$ to cover a nonempty $I(M)$.

Theorem 2. *$(\mathcal{H}, \mathcal{G})$ is learnable in the PLEX framework if it has polynomial width and $\mathcal{H}$ is well-structured, has polynomial height, and is adequate for $\mathcal{G}$.*

Proof. Algorithm 2 is a general schema for action model learning in the PLEX framework. The current model M is initialized to the optimistic model, which must exist as $\mathcal{H}$ is well-structured. Given M and $\mathcal{G}$, ExperimentDesign computes a polynomial-sized cover G. If G is empty, then the model cannot be refined further; otherwise, given M and a goal $g \in G$, Planner may signal failure if either no state satisfies g or states satisfying g are not reachable from the current state of the simulator. If Planner signals failure on all of the goals, then none of the informative states are reachable and M cannot be refined further. If Planner does output a plan, then ModelLearner either refines M somewhere along the plan execution or it refines M by executing an informative action after reaching a state that satisfies g. The existence of an adequate model is assured in the original hypothesis space and there is no risk of losing such model by removing illegal transitions. A new cover is

Algorithm 2 PLEX LEARNING SCHEMA
Input: Initial state s, hypothesis space $\mathcal{H}$, goal language $\mathcal{G}$
Output: Model M

```
1: $M \leftarrow$ optimistic model of $\mathcal{H}$
2: loop
3:     $G \leftarrow$ EXPERIMENTDESIGN$(M, \mathcal{G})$
4:     if $G = \varnothing$ then
5:         return $M$
6:     for $g \in G$ do
7:         $plan \leftarrow$ PLANNER$(M, (s, g))$
8:         if $plan \neq$ failure then
9:             break
10:    if $plan$ = failure then
11:        return $M$
12:    for $a$ in $plan$ do
13:        $s' \leftarrow$ SIMULATOR$(a)$
14:        $M \leftarrow$ MODELLEARNER$(M, (s, a, s'))$
15:        $s \leftarrow s'$
16:        if $M$ has been updated then
17:            break
18:    if $M$ has not been updated then
19:        $a \leftarrow$ an element in INFORMATIVEACTIONS$(M, s)$
20:        $s' \leftarrow$ SIMULATOR$(a)$
21:        $M \leftarrow$ MODELLEARNER$(M, (s, a, s'))$
22:        $s \leftarrow s'$
23: return $M$
```

computed every time M is refined, and the process continues until all experiments are exhausted. As the number of successful plans is bounded by the height h of $\mathcal{H}$ and the number of failures per successful plan is bounded by a polynomial in the width w of $(\mathcal{H}, \mathcal{G})$, the total number of calls to PLANNER is $O(h \cdot \text{poly}(w))$, which is a polynomial in the domain parameters. Thus, $(\mathcal{H}, \mathcal{G})$ is learnable in the PLEX framework. □

Definition 19. $(\mathcal{H}, \mathcal{G})$ *permits* efficient *experiment design if, for any $M \in \mathcal{H}$, ① there exists an algorithm that outputs a polynomial-sized cover of $I(M)$ in polynomial time and ② there exists an algorithm that outputs the set of informative actions in M for any state in polynomial time.*

Note that if $(\mathcal{H}, \mathcal{G})$ permits efficient experiment design, then has polynomial width because no algorithm can always guarantee to output a polynomial-sized cover otherwise.

Corollary 2. *$(\mathcal{H}, \mathcal{G})$ is polynomial-time learnable in the PLEX framework if it permits efficient experiment design and $\mathcal{H}$ is efficiently well-structured and has polynomial height.*

Proof. If $(\mathcal{H}, \mathcal{G})$ permits efficient experiment design, then a cover can be computed in polynomial time. As $\mathcal{H}$ is efficiently well-structured, MODELLEARNER can take the current model and an observation and return the updated model in polynomial time. From the proof of Theorem 2 and the fact that the innermost loop of Algorithm 2 is bounded by the longest length l of a plan and picking an informative action can be done efficiently, we can deduce that its computational complexity is $O(h \cdot \text{poly}(w) \cdot (l + \text{poly}(n, |A|, |D|))$, which is a polynomial in the domain parameters and l. Thus, assuming that all the other components run in $O(1)$ time, $(\mathcal{H}, \mathcal{G})$ is polynomial-time learnable in the PLEX framework. □

The key differences between the MBP and PLEX frameworks are highlighted in Table 1.

Sets of Action Productions

This section describes a concrete representational class for action models — sets of action productions — and proves its learnability in the MBP and PLEX frameworks. For brevity, let $d = |D|$ and $m = |A|$.

An *action production* r is defined as "act : pre $\to$ post" where act(r) is an action and the precondition pre(r) and postcondition post(r) are conjunctions of "variable = value" literals.

Definition 20. *A production r is* triggered *by a transition (s, a, s') if s satisfies the precondition pre(r) and a = act(r). A production r is* (weakly) consistent *with (s, a, s') if either ① r is not triggered by (s, a, s') or ② s' satisfies the post(r) and all variables not mentioned in post(r) have the same values in both s and s'.*

An example of an action production is "Do : $v_1 = 0, v_2 = 1 \to v_1 = 2, v_3 = 1$". It is triggered only when the Do action is executed in a state where $v_1 = 0$ and $v_2 = 1$, and defines the value of v_1 to be 2 and v_3 to be 1 in the next state, with all other variables staying unchanged.

A set of action productions (SAP) is consistent with a state transition if all productions in the SAP are consistent with it. Let k-SAP be the hypothesis space of models represented by a SAP with no more than k variables per production. Note that k-SAP is strictly more general than propositional STRIPS operators since it can express conditional effects, where each conditional effect might depend on a different set of variables.

Lemma 2. *k-SAP is efficiently well-structured.*

Proof. k-SAP is closed under union because unioning the productions of any two SAPs results in a SAP, which implies that it is well-structured. Given an observed transition (s, a, s'), a k-SAP model is refined by removing productions that are not consistent with (s, a, s'), which takes polynomial time. □

Lemma 3. *k-SAP has polynomial height.*

Proof. The total number of productions in k-SAP $= O\big(m \sum_{i=1}^{k} \binom{n}{i} (d+1)^{2i}\big) = O(mn^k d^{2k})$ because a production can have one of m actions and up to k relevant variables figuring on either side of the production, each variable set to a value in its domain. At the root of the generalization graph is the hypothesis that contains all the productions, and at the leaf is the hypothesis that contains no productions. Because the longest path from the root to the leaf involves removing a single production at a time, the height of k-SAP is $O(mn^k d^{2k})$. □

Theorem 3. *k-SAP is polynomial-time learnable in the MBP framework.*

Table 1: The principal differences between the MBP and PLEX frameworks.

	MBP	**PLEX**
Planning problem	Externally generated	Internally generated
Experiment design	Irrelevant	Relevant
Sample complexity	Polynomial number of mistakes	Polynomial number of planning attempts
Computational complexity	Polynomial per response	Polynomial

Proof. This follows from Lemmas 2 and 3, and Corollary 1. □

A k-SAP model is nondeterministic if it contains two productions for the same action whose preconditions overlap but postconditions disagree. This ambiguity can be resolved by picking any state that triggers both productions and executing the corresponding action. Let the goal language *Conj* consist of all goals that can be expressed as conjunctions of "variable = value" constraints.

Lemma 4. *(k-SAP, Conj) permits efficient experiment design.*

Proof. Given an action, the possible pairs of overlapping productions in k-SAP is $O(n^{2k}d^{4k})$. Each pair gives rise to exactly one goal described by the conjunctive union of preconditions of the two productions. Hence, the width of (k-SAP, *Conj*) is $O(n^{2k}d^{4k})$, which is a polynomial in the domain parameters. Consequently, experiment design is efficient because it involves searching a polynomial number of pairs for those with overlapping preconditions and conflicting postconditions. □

Theorem 4. *(k-SAP, Conj) is polynomial-time learnable in the PLEX framework.*

Proof. This follows from Lemmas 2, 3, and 4, and Corollary 2. □

Discussion and Related Work

The first contribution of this work is the identification of the role of adequate models in characterizing the complexity of learning with a small number of mistakes. Exact action models are sometimes too complex for the purposes of planning adequately and require too much effort to learn. The frameworks allow the agent to learn models that are adequate for planning. The second contribution is the development of the PLEX framework which allows the learner to direct its exploration in ways that inform its model. We clarify the relationship between the expressiveness of the goal language and its usefulness in learning the action models. The third contribution is providing specific algorithms for learning a concrete hypothesis space that is in some ways more general than standard action modeling languages. For example, unlike propositional STRIPS operators, k-SAP captures the conditional effects of actions.

Our work is partly inspired by the exploration problem in model-based reinforcement learning in factored MDPs. Here one seeks a PAC-MDP algorithm, which guarantees that the agent is performing suboptimally for at most a polynomial number of time steps in the sizes of the state and the action spaces (Strehl et al. 2006). RMAX is an example of PAC-MDP algorithm which employs the principle of optimism under uncertainty to explore efficiently (Brafman and Tennenholtz 2002). It initiates learning with optimistic transition and reward models that assumes that the unknown states have high rewards, which automatically encourages the agent to visit these states. Unfortunately, interesting MDPs have prohibitively large state spaces and being polynomial in their size is not good enough. DBN-E^3 and Factored-RMAX learn action models represented as dynamic Bayesian networks (DBNs) and guarantee at most a polynomial number of suboptimal actions in the minimal size of their domain models (Kearns and Koller 1999; Guestrin, Patrascu, and Schuurmans 2002). A generalization of this approach to arbitrary model classes is based on the notion of KWIK learning (Li, Littman, and Walsh 2008). KWIK-learning is a function learning framework that extends PAC-learning by requiring that the learner knows exactly when its knowledge of the target function is approximately correct. KWIK-learning of action models can be plugged into RMAX to yield KWIK-RMAX, which guarantees polynomial scaling with respect to the size of the action models. The key idea is to run RMAX in the outer loop and generate useful new experience for the internal KWIK learner. When the KWIK learner reports that it does not know a particular transition, RMAX assumes a transition to a high reward state, biasing KWIK-RMAX toward exploring such states.

The MDP framework is more general than the deterministic action models considered here in that it includes stochasticity and rewards. However, the following reasons motivate the study of deterministic models. First, note that all these frameworks (including ours) leave open the problem of probabilistic planning, which is much harder and lesser understood than deterministic planning. Second, studying deterministic models offers some important insights. For example, we have uncovered the notions of adequacy and well-structured hypothesis spaces which are central for successful learning in our framework. The well-structured property is related to the well-ordered property of Natarajan (1987), which is a necessary and sufficient condition for concept learning with one-sided error. It is thus possible to view our work as reducing model-learning to one-sided mistake-bounded concept learning, where the learned hypothesis is always guaranteed to be a superset of the target concept. Without the one-sided mistake guarantee, the learner might not be able to plan successfully in some cases because its hypothesis may not allow certain transitions that are needed for a successful plan. To get around this difficulty, Walsh (2010) studies the efficient learning in the framework of apprenticeship learning where a teacher gives examples of better plans when the learner produces a bad plan. The KWIK framework can be viewed as another way of getting around this difficulty, where the learner explicitly signals when its model is inadequate. Third, studying goal-directed planning allows us to explicate the structural interplay between the action model language and the goal language, an issue that does not arise in the MDP framework.

As for the learning of relational planning operators, Opmaker in the GIPO system (McCluskey, Richardson, and Simpson 2002) takes as input a partial domain knowledge of the object behaviors and descriptions, training operator sequences, and user interaction and outputs parameterized flat or hierarchical operator models. In contrast, our learning schemas facilitate autonomous operator learning for propositional descriptions of the primitive operators. The ARMS algorithm (Yang, Wu, and Jiang 2005) learns approximate operator models from successful example plans (without assuming that the intermediate states are provided) by gathering knowledge on the statistical distribution of frequent sets of actions in the example plans and solving a weighted satisfiability problem. Instead, our learning schemas assume full observability and are online in nature.

While STRIPS-like languages served us well in planning research by creating a common useful platform, they are not designed from the point of view of learnability or planning efficiency. Many domains such as robotics and real-time strategy games are not amenable to such clean and simple action specification languages. This suggests an approach where the learner considers increasingly complex models as dictated by its planning needs. For example, the model learner might consider increasing the value of k if the parameterized hypothesis spaces like k-SAP are inadequate for the goals encountered. In general, this motivates for a more comprehensive framework in which planning and learning are tightly integrated, the premise of this paper. Another direction is to investigate better exploration methods that go beyond using optimistic models to include Bayesian and utility-guided optimal exploration.

Acknowledgments

This research is supported by the Army Research Office under grant number W911NF-09-1-0153. We thank the reviewers for many useful comments. We also thank Roni Khardon for pointing out some crucial errors in a previous version of the paper and the organizers of the workshop for their patience.

References

Brafman, R., and Tennenholtz, M. 2002. R-MAX — A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research* 3:213–231.

Guestrin, C.; Patrascu, R.; and Schuurmans, D. 2002. Algorithm-Directed Exploration for Model-Based Reinforcement Learning in Factored MDPs. In *ICML*.

Kearns, M., and Koller, D. 1999. Efficient Reinforcement Learning in Factored MDPs. In *IJCAI*.

Li, L.; Littman, M.; and Walsh, T. 2008. Knows What It Knows: A Framework for Self-Aware Learning. In *ICML*.

Marthi, B.; Russell, S.; and Wolfe, J. 2007. Angelic Semantics for High-Level Actions. In *ICAPS*.

McCluskey, T.; Richardson, N.; and Simpson, R. 2002. An Interactive Method of Inducing Operator Descriptions. In *International Conference on Artificial Intelligence Planning Systems*.

Natarajan, B. K. 1987. On Learning Boolean Functions. In *Annual ACM Symposium on Theory of Computing*.

Strehl, A.; Li, L.; Wiewiora, E.; Langford, J.; and Littman, M. 2006. PAC Model-free Reinforcement Learning. In *ICML*.

Walsh, T., and Littman, M. 2008. Efficient Learning of Action Schemas and Web-Service Descriptions. In *AAAI*.

Walsh, T. 2010. *Efficient Learning of Relational Models for Sequential Decision Making*. Ph.D. Dissertation, Rutgers University.

Yang, Q.; Wu, K.; and Jiang, Y. 2005. Learning Action Models from Plan Examples with Incomplete Knowledge. In *International Conference on Automated Planning and Scheduling*.