Learning Action Models from State Observations

#1186

Abstract

This paper presents a classical planning compilation for learning STRIPS action models from state observations. The compilation approach does not require observing the precise actions that produced the observed states because such actions, are determined by a planner. Furthermore, the compilation is extensible to assess how well a STRIPS action model matches a given set of observations. Last but not least, the paper evaluates the performance of the compilation approach by learning action models for a wide range of classical planning domains from the International Planning Competition (IPC) and assessing the learned models with respect to (1), test sets of state observations and (2), the true models.

1 Introduction

Learning action models is a promising approach to relief the *knowledge acquisition bottleneck* of automated planning [Kambhampati, 2007]. Over the last decade, the learning of planning action models has been addressed with various approaches. ARMS [Yang *et al.*, 2007] receives plan traces, as training samples, and defines the learning task as a set of weighted constraints that the input plans must hold, which is then solved by a MAX-SAT solver. ARMS validates the learned models with test sets of example plans from six different IPC domains. The learning algorithm of SLAF [Amir and Chang, 2008] computes a CNF formula which is consistent with the input plan traces and partially observed states. SLAF assesses the quality of the learned models with respect to the corresponding reference model. In the case of LOCM [Cresswell *et al.*, 2013], the learning samples are only examples plans, without requiring any information about predicates or states. LOCM relies on assumptions on the kind of domain structure of the model and compares the learned models with the corresponding reference model [Gregory and Cresswell, 2016]. Finally, AMAN [Zhuo and Kambhampati, 2013] can work with incorrectly observed plan traces and evaluates the learned models in three IPC domains, regarding the corresponding reference model.

The learning samples required by all the aforementioned approaches are observations of plan executions, in the form of plan traces, and all of them, but ARMS, evaluate learning performance by syntactically comparing the learned models with respect to the corresponding reference model.

Motivated by recent advances on the synthesis of generative models with classical planning [Bonet *et al.*, 2009; Segovia-Aguas *et al.*, 2016; 2017], we present a novel planning compilation for learning STRIPS action models from state observations. A solution to the classical planning task that results from our compilation is a sequence of actions that determines the learned action model, i.e. the preconditions and effects of the target STRIPS operator schemas. **Learning action models as planning** leverages off-the-shelf planners and opens up a way towards *bootstrapping* planning action models, enabling a planner to gradually learn/update its action model. The practicality of the compilation approach allow us to report learning results over fifteen IPC planning domains.

For the training samples, we adopt a middle ground between unstructured inputs and plan traces, wherein only state observations are required. **Learning from state observations** is a relevant advancement as, in many applications, the actual actions executed by the observed agent are not available but, instead, the resulting states can be observed. Learning action models from state observations broadens the range of application to external observers and facilitates the representation of imperfect observability (as shown in plan recognition [Sohrabi *et al.*, 2016]) as well as learning from unstructured data (like state images [Asai and Fugunaga, 2018]).

Finally, our approach is able to **assess how well a STRIPS action model matches a given set of observations**. Our compilation is extensible to accept a learned model as input besides the state observations. This extension allows us to transform the input model into a new model that induces the observations whilst assessing the amount of edition required by the input model to induce the given observations. The empirical evaluation of our learning approach is two-fold: First the learned STRIPS action models are tested with a set of state observation sequences and second, the learned models are compared to the corresponding reference model.

2 Background

Our approach for learning STRIPS action schemas from state observations is compiling this learning task into a classical planning task with conditional effects.

2.1 Classical planning with conditional effects

We use F to denote the set of *fluents* (propositional variables) describing a state. A *literal* l is a valuation of a fluent $f \in F$, i.e. either $l = f$ or $l = \neg f$. A set of literals L represents a partial assignment of values to fluents (without loss of generality, we will assume that L does not assign conflicting values to any fluent). We use $\mathcal{L}(F)$ to denote the set of all literal sets on F, i.e. all partial assignments of values to fluents.

A *state* s is a full assignment of values to fluents, i.e. $|s| = |F|$, so the size of the state space is $2^{|F|}$. Explicitly including negative literals $\neg f$ in states simplifies subsequent definitions but often, we will abuse notation by defining a state s only in terms of the fluents that are true in s, as is common in STRIPS planning.

A *classical planning frame* is a tuple $\Phi = \langle F, A \rangle$, where F is a set of fluents and A is a set of actions. An action $a \in A$ is defined with *preconditions*, $\mathsf{pre}(a) \subseteq \mathcal{L}(F)$, *positive effects*, $\mathsf{eff}^+(a) \subseteq \mathcal{L}(F)$, and *negative effects* $\mathsf{eff}^-(a) \subseteq \mathcal{L}(F)$. We say that an action $a \in A$ is *applicable* in a state s iff $\mathsf{pre}(a) \subseteq s$. The result of applying a in s is the *successor state* denoted by $\theta(s, a) = \{s \setminus \mathsf{eff}^-(a)) \cup \mathsf{eff}^+(a)\}$.

An action $a \in A$ with conditional effects is defined as a set of *preconditions* $\mathsf{pre}(a) \in \mathcal{L}(F)$ and a set of *conditional effects* $\mathsf{cond}(a)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a)$ is composed of two sets of literals $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*. An action $a \in A$ is *applicable* in a state s if and only if $\mathsf{pre}(a) \subseteq s$, and the *triggered effects* resulting from the action application are the effects whose conditions hold in s:

$$triggered(s, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a), C \subseteq s} E,$$

The result of applying action a in state s is the *successor* state $\theta(s, a) = \{s \setminus \mathsf{eff}_c^-(s, a)) \cup \mathsf{eff}_c^+(s, a)\}$ where $\mathsf{eff}_c^-(s, a) \subseteq triggered(s, a)$ and $\mathsf{eff}_c^+(s, a) \subseteq triggered(s, a)$ are, respectively, the triggered *negative* and *positive* effects.

A *classical planning problem* is a tuple $P = \langle F, A, I, G \rangle$, where I is an initial state and $G \subseteq \mathcal{L}(F)$ is a goal condition. A *plan* for P is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that induces the *state trajectory* $\langle s_0, s_1, \ldots, s_n \rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, a_i is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$. The *plan length* is denoted with $|\pi| = n$. A plan π *solves* P iff $G \subseteq s_n$, i.e. if the goal condition is satisfied at the last state reached after following the application of the plan π in the initial state I.

2.2 STRIPS action schemas

This work addresses the learning of PDDL action schemas that follow the STRIPS requirement [McDermott *et al.*, 1998; Fox and Long, 2003]. Figure 1 shows the *stack* action schema, coded in PDDL, from a four-operator *blocksworld* [Slaney and Thiébaux, 2001].

To formalize the output of the learning task, we assume that fluents F are instantiated from a set of *predicates* Ψ, as in PDDL. Each predicate $p \in \Psi$ has an argument list of arity $ar(p)$. Given a set of *objects* Ω, the set of fluents F is induced by assigning objects in Ω to the arguments of predicates in Ψ,

```
(:action stack
 :parameters (?v1 ?v2 - object)
 :precondition (and (holding ?v1) (clear ?v2))
 :effect (and (not (holding ?v1))
              (not (clear ?v2))
              (handempty) (clear ?v1)
              (on ?v1 ?v2)))
```

Figure 1: STRIPS operator schema coding, in PDDL, the *stack* action from a four-operator *blocksworld*.

i.e. $F = \{p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}\}$ s.t. Ω^k is the k-th Cartesian power of Ω.

Let $\Omega_v = \{v_i\}_{i=1}^{\max_{a \in A} ar(a)}$ be a new set of objects ($\Omega \cap \Omega_v = \emptyset$), denoted as *variable names*, and that is bound by the maximum arity of an action in a given planning frame. For instance, in a three-block *blocksworld* $\Omega = \{block_1, block_2, block_3\}$ while $\Omega_v = \{v_1, v_2\}$ because the operators with the maximum arity, `stack` and `unstack`, have arity two. We define F_v, a new set of fluents s.t. $F \cap F_v = \emptyset$, that results from instantiating Ψ using only the objects in Ω_v and defines the elements that can appear in an action schema. For the *blocksworld*, F_v={`handempty`, `holding(`v_1`)`, `holding(`v_2`)`, `clear(`v_1`)`, `clear(`v_2`)`, `ontable(`v_1`)`, `ontable(`v_2`)`, `on(`v_1, v_1`)`, `on(`v_1, v_2`)`, `on(`v_2, v_1`)`, `on(`v_2, v_2`)`}.

We assume also that actions $a \in A$ are instantiated from STRIPS operator schemas $\xi = \langle head(\xi), pre(\xi), add(\xi), del(\xi) \rangle$ where:

- $head(\xi) = \langle name(\xi), pars(\xi) \rangle$, is the operator *header* defined by its name and the corresponding *variable names*, $pars(\xi) = \{v_i\}_{i=1}^{ar(\xi)}$. The headers of a four-operator *blocksworld* are `pickup(`v_1`)`, `putdown(`v_1`)`, `stack(`v_1, v_2`)` and `unstack(`v_1, v_2`)`.
- The preconditions $pre(\xi) \subseteq F_v$, the negative effects $del(\xi) \subseteq F_v$, and the positive effects $add(\xi) \subseteq F_v$ such that, $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$.

Finally, given the set of predicates Ψ and the header of a STRIPS operator schema ξ, we define $F_v(\xi) \subseteq F_v$ as the subset of elements that can appear in the action schema ξ and that confine its space of possible action models. For instance, for the *stack* action schema F_v(`stack`) $= F_v$ while F_v(`pickup`)={`handempty`, `holding(`v_1`)`, `clear(`v_1`)`, `ontable(`v_1`)`, `on(`v_1, v_1`)`} excludes the fluents from F_v that involve v_2 because the action header `pickup(`v_1`)` contains the single parameter v_1.

3 Learning STRIPS action models

This paper addresses the learning task that corresponds to observing an agent acting in the world but watching only the results of its plan executions, the actual executed actions are unobserved. This learning task is defined as $\Lambda = \langle \mathcal{M}, \Psi, \mathcal{O} \rangle$:

- $\mathcal{M}$ is the set of *empty* operator schemas, wherein each $\xi \in \mathcal{M}$ is only composed of $head(\xi)$.
- Ψ is the set of predicates, that define the abstract state space of a given classical planning frame.

```
;;;;;; Headers in M
(pickup v1) (putdown v1)
(stack v1 v2} (unstack v1 v2)

;;;;;; Predicates Ψ
(handempty) (holding ?o  - object)
(clear ?o - object) (ontable ?o - object)
(on ?o1 - object ?o2 - object)

;;;;;; Observations O
;;; observation #0
(clear block2) (on block2 block1)
(ontable block1) (handempty)

;;; observation #1
(holding block2) (clear block1) (ontable block1)

;;; observation #2
(clear block1) (ontable block1)
(clear block2) (ontable block2) (handempty)

;;; observation #3
(holding block1) (clear block2) (ontable block2)

;;; observation #4
(clear block1) (on block1 block2)
(ontable block2) (handempty)
```

Figure 2: Example of a Λ task for learning a STRIPS action model in the *blocksworld* from a sequence of five state observations.

- $\mathcal{O} = \langle s_0, s_1, \ldots, s_n \rangle$ is a sequence of *state observations* obtained observing the execution of an *unobserved* plan $\pi = \langle a_1, \ldots, a_n \rangle$.

A solution to Λ is a set of operator schema $\mathcal{M}'$ compliant with the headers in $\mathcal{M}$, the predicates Ψ, and the state observation sequence $\mathcal{O}$. Figure 2 shows a Λ task for learning the *blocksworld* STRIPS action model from the five-state observations sequence that corresponds to inverting a 2-block tower.

3.1 Learning with classical planning

Our approach for addressing a Λ learning task is compiling it into a classical planning task P_Λ with conditional effects. A planning compilation is a suitable approach for addressing Λ because a solution must not only determine the STRIPS action model $\mathcal{M}'$ but also, the *unobserved* plan $\pi = \langle a_1, \ldots, a_n \rangle$, that explains $\mathcal{O}$. The intuition behind the compilation is that a solution to the resulting classical planning task is a sequence of actions that:

1. **Programs the action model $\mathcal{M}'$.** A solution plan starts with a *prefix* that, for each $\xi \in \mathcal{M}$, determines which fluents $f \in F_v(\xi)$ belong to its $pre(\xi)$, $del(\xi)$ and $add(\xi)$ sets.
2. **Validates the action model $\mathcal{M}'$ in $\mathcal{O}$.** The solution plan continues with a postfix that produces the given sequence of states $\langle s_0, s_1, \ldots, s_n \rangle$ using the programmed action model $\mathcal{M}'$.

Given a learning task $\Lambda = \langle \mathcal{M}, \Psi, \mathcal{O} \rangle$ the compilation outputs a classical planning task $P_\Lambda = \langle F_\Lambda, A_\Lambda, I_\Lambda, G_\Lambda \rangle$:

- F_Λ contains:
  - The set of fluents F built instantiating the predicates Ψ with the objects appearing in the input observations $\mathcal{O}$, i.e. `block1` and `block2` in Figure 2.
  - Fluents $pre_f(\xi)$, $del_f(\xi)$ and $add_f(\xi)$, for every $f \in F_v(\xi)$, that represent the programmed action model. If a fluent $pre_f(\xi)/del_f(\xi)/add_f(\xi)$ holds, it means that f is a precondition/negative/positive effect in the schema $\xi \in \mathcal{M}'$. For instance, the preconditions of the $stack$ schema (Figure 1) are represented by the pair of fluents `pre_holding_stack_`v_1 and `pre_clear_stack_`v_2 set to *True*.
  - The fluents $mode_{prog}$ and $mode_{val}$ indicating whether the operator schemas are programmed or validated and the fluents $\{test_i\}_{1 \leq i \leq n}$, indicating the observation where the action model is validated.
- I_Λ contains the fluents from F that encode s_0 (the first observation) and $mode_{prog}$ set to true. Our compilation assumes that initially, operator schemas are programmed with every possible precondition (the most specific learning hypothesis), no negative effect and no positive effect. Therefore fluents $pre_f(\xi)$, for every $f \in F_v(\xi)$, hold also at the initial state.
- $G_\Lambda = \bigcup_{1 \leq i \leq n} \{test_i\}$, requires that the programmed action model is validated in all the input observations.
- A_Λ comprises three kinds of actions:
  1. Actions for *programming* operator schema $\xi \in \mathcal{M}$:
     - Actions for **removing** a *precondition* $f \in F_v(\xi)$ from the action schema $\xi \in \mathcal{M}$.

$$\begin{aligned} \mathsf{pre}(\mathsf{programPre}_{\mathsf{f},\xi}) =& \{\neg del_f(\xi), \neg add_f(\xi), \\ & mode_{prog}, pre_f(\xi)\}, \\ \mathsf{cond}(\mathsf{programPre}_{\mathsf{f},\xi}) =& \{\emptyset\} \rhd \{\neg pre_f(\xi)\}. \end{aligned}$$

     - Actions for **adding** a *negative* or *positive* effect $f \in F_v(\xi)$ to the action schema $\xi \in \mathcal{M}$.

$$\begin{aligned} \mathsf{pre}(\mathsf{programEff}_{\mathsf{f},\xi}) =& \{\neg del_f(\xi), \neg add_f(\xi), \\ & mode_{prog}\}, \\ \mathsf{cond}(\mathsf{programEff}_{\mathsf{f},\xi}) =& \{pre_f(\xi)\} \rhd \{del_f(\xi)\}, \\ & \{\neg pre_f(\xi)\} \rhd \{add_f(\xi)\}. \end{aligned}$$

  2. Actions for *applying* a programmed operator schema $\xi \in \mathcal{M}$ bound with objects $\omega \subseteq \Omega^{ar(\xi)}$. Given that the operators headers are known, the variables $pars(\xi)$ are bound to the objects in ω that appear at the same position.

$$\begin{aligned} \mathsf{pre}(\mathsf{apply}_{\xi,\omega}) =& \{pre_f(\xi) \implies p(\omega)\}_{\forall p \in \Psi, f = p(pars(\xi))} \\ & \cup \{\neg mode_{val}\}, \\ \mathsf{cond}(\mathsf{apply}_{\xi,\omega}) =& \{del_f(\xi)\} \rhd \{\neg p(\omega)\}_{\forall p \in \Psi, f = p(pars(\xi))}, \\ & \{add_f(\xi)\} \rhd \{p(\omega)\}_{\forall p \in \Psi, f = p(pars(\xi))}, \\ & \{mode_{prog}\} \rhd \{\neg mode_{prog}\}, \\ & \{\emptyset\} \rhd \{mode_{val}\}. \end{aligned}$$

3. Actions for *validating* an observation $1 \leq i \leq n$.

$$\mathsf{pre}(\mathsf{validate_i}) = s_i \cup \{test_j\}_{j \in 1 \leq j < i} \cup \{\neg test_j\}_{j \in i \leq j \leq n} \cup \{mode_{val}\},$$
$$\mathsf{cond}(\mathsf{validate_i}) = \{\emptyset\} \rhd \{test_i, \neg mode_{val}\}.$$

3.2 Compilation properties

Lemma 1. *Soundness. Any classical plan π that solves P_Λ induces an action model $\mathcal{M}'$ that solves $\Lambda = \langle \mathcal{M}, \Psi, \mathcal{O} \rangle$.*

Proof sketch. Once operator schemas $\mathcal{M}'$ are programmed, they can only be applied and validated, because of the $mode_{prog}$ fluent. In addition, P_Λ is only solvable if fluents $\{test_i\}$, $1 \leq i \leq n$ hold at the last reached state. These goals can only be achieved executing an applicable sequence of programmed operator schemas that reaches every state $s_i \in \mathcal{O}$, starting from s_0 and following the sequence $1 \leq i \leq n$. This means that the programmed action model $\mathcal{M}'$ complies with the provided observations $\mathcal{O}$ and hence, solves Λ. □

Lemma 2. *Completeness. Any* STRIPS *action model $\mathcal{M}'$ that solves a $\Lambda = \langle \mathcal{M}, \Psi, \mathcal{O} \rangle$ learning task, is computable solving the corresponding classical planning task P_Λ.*

Proof sketch. By definition, $F_v(\xi) \subseteq F_\Lambda$ fully captures the full set of elements that can appear in a STRIPS action schema $\xi \in \mathcal{M}$ given its header and the set of predicates Ψ. The compilation does not discard any possible STRIPS action schema definable within F_v that satisfies the state trajectory constraint given by $\mathcal{O}$. □

The size of the classical planning task P_Λ output by the compilation depends on:

- The arity of the actions headers in $\mathcal{M}$ and the predicates Ψ that are given as input to the Λ learning task. The larger these numbers, the larger the $F_v(\xi)$ sets, that define the $pre_f(\xi)/del_f(\xi)/add_f(\xi)$ fluents set and the corresponding set of *programming* actions.
- The number of given state observations. The larger $|\mathcal{O}|$, the more $test_i$ fluents and $\mathsf{validate_i}$ actions in P_Λ.

4 Evaluating STRIPS action models

We assess how well a STRIPS action model $\mathcal{M}$ explains a given sequence of observations $\mathcal{O}$ according to the amount of *edition* required by $\mathcal{M}$ to induce $\mathcal{O}$.

4.1 Edition of STRIPS action models

We first define the allowed *operations* to edit a given STRIPS action model. With the aim of keeping a tractable branching factor of the planning instances that results from our compilation, we only define two *edit operations*:

- *Deletion.* A fluent $pre_f(\xi)/del_f(\xi)/add_f(\xi)$ is removed from the operator schema $\xi \in \mathcal{M}$, $f \in F_v(\xi)$.
- *Insertion.* A fluent $pre_f(\xi)/del_f(\xi)/add_f(\xi)$ is added to the operator schema $\xi \in \mathcal{M}$, $f \in F_v(\xi)$.

We can now formalize an edit distance that quantifies how dissimilar two given STRIPS action models are. The distance is symmetric and meets the *metric axioms* provided that the two *edit operations* have the same positive cost.

Definition 3. *Let $\mathcal{M}$ and $\mathcal{M}'$ be two* STRIPS *action models, both built from the same set of possible elements F_v. The* **edit distance***, denoted as $\delta(\mathcal{M}, \mathcal{M}')$, is the minimum number of* edit operations *required to transform $\mathcal{M}$ into $\mathcal{M}'$.*

Since F_v is a bound set, the maximum number of edits that can be introduced to a given action model defined within F_v is bound as well. In more detail, for an operator schema $\xi \in \mathcal{M}$ the maximum number of edits that can be introduced to their precondition set is $|F_v(\xi)|$ while the max number of edits that can be introduced to the effects is twice $|F_v(\xi)|$.

Definition 4. *The* ***maximum edit distance*** *of an* STRIPS *action model $\mathcal{M}$ built from the set of possible elements F_v is $\delta(\mathcal{M}, *) = \sum_{\xi \in \mathcal{M}} 3|F_v(\xi)|$.*

4.2 The observation edit distance

We define now an edit distance to asses the quality of a learned action model with respect to a sequence of state observations.

Definition 5. *Given an action model $\mathcal{M}$, built from F_v, and the observations sequence $\mathcal{O} = \langle s_0, s_1, \ldots, s_n \rangle$. The* **observation edit distance***, denoted by $\delta(\mathcal{M}, \mathcal{O})$, is the minimal edit distance from $\mathcal{M}$ to any model $\mathcal{M}'$, defined also within F_v, such that $\mathcal{M}'$ can produce a plan $\pi = \langle a_1, \ldots, a_n \rangle$ that induces $\mathcal{O}$;*

$$\delta(\mathcal{M}, \mathcal{O}) = \min_{\forall \mathcal{M}' \to \mathcal{O}} \delta(\mathcal{M}, \mathcal{M}')$$

The Λ learning tasks can swap the roles of two operators whose headers match or two action parameters that belong to the same type (e.g. the *blocksworld* operator `stack` can be *learned* with the preconditions and effects of the `unstack` operator and vice versa, or the parameters of the `stack` operator can be swapped). The *observation edit distance* is a semantic measure that is robust to role changes of this kind.

Unlike the error function defined by ARMS [Yang *et al.*, 2007], the *observation edit distance* assesses, with a single expression, the flaws in the preconditions and effects of a given learned model. This fact enables the recognition of STRIPS action models. The idea, taken from *plan recognition as planning* [Ramírez and Geffner, 2009], is to map distances into likelihoods. The *observation edit distance* could be mapped into a likelihood with the following expression $P(\mathcal{O}|\mathcal{M}) = 1 - \frac{\delta(\mathcal{M},\mathcal{O})}{\delta(\mathcal{M},*)}$.

4.3 Computing the observation edit distance

Our compilation is extensible to compute the *observation edit distance* by simply considering that the model $\mathcal{M}$, given in $\Lambda = \langle \mathcal{M}, \Psi, \mathcal{O} \rangle$, is *non-empty*. In other words, now $\mathcal{M}$ is a set of given operator schemas, wherein each $\xi \in \mathcal{M}$ initially contains $head(\xi)$ but also the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ sets. A solution to the planning task resulting from the extended compilation is a sequence of actions that:

1. **Edits the action model $\mathcal{M}$.** A solution plan starts with a *prefix* that modifies the preconditions and effects of the action schemas in $\mathcal{M}$ using to the two *edit operations*.
2. **Validates the edited model $\mathcal{M}'$ in $\mathcal{O}$.** The solution plan continues with a postfix that validates the edited model on the given observations $\mathcal{O}$, as in Section 3.

```
00 :  (insert_add_handempty_stack)
01 :  (insert_add_clear_stack_var1)
02 :  (apply_unstack block2 block1)
03 :  (validate_1)
04 :  (apply_putdown block2)
05 :  (validate_2)
06 :  (apply_pickup block1)
07 :  (validate_3)
08 :  (apply_stack block1 block2)
09 :  (validate_4)
```

Figure 3: Plan for editing a given *blockswold* schema and validating it at the state observations shown in Figure 2.

Now Λ is not a learning task but the task of editing $\mathcal{M}$ to produce the observations $\mathcal{O}$, which results in the edited model $\mathcal{M}'$. The output of the extended compilation is a classical planning task $P'_\Lambda = \langle F_\Lambda, A'_\Lambda, I'_\Lambda, G_\Lambda \rangle$:

- F_Λ and G_Λ are defined as in the previous compilation.
- I'_Λ contains the fluents from F that encode s_0 and $mode_{prog}$ set to true. In addition, the input action model $\mathcal{M}$ is now encoded in the initial state. This means that the fluents $pre_f(\xi)/del_f(\xi)/add_f(\xi)$, $f \in F_v(\xi)$, hold in the initial state iff they appear in $\mathcal{M}$.
- A'_Λ, comprises the same three kinds of actions of A_Λ. The actions for *applying* an already programmed operator schema and the actions for *validating* an observation are defined exactly as in the previous compilation. The only difference here is that the actions for *programming* the operator schema now implement the two *edit operations* (i.e. include actions for *inserting* a precondition and for *deleting* a negative/positive effect).

To illustrate this, the plan of Figure 3 solves the classical planning task P'_Λ that corresponds to editing a *blocksworld* action model where the positive effects `(handempty)` and `(clear ?v1)` of the `stack` schema are missing. The plan edits first the `stack` schema, *inserting* these two positive effects, and then validates the edited action model in the five-observation sequence of Figure 2.

Our interest when computing the *observation edit distance* is not in $\mathcal{M}'$ but in the number of required *edit operations* for that $\mathcal{M}'$ is validated in the given observations, e.g. $\delta(\mathcal{M}, \mathcal{O}) = 2$ for the example in Figure 3. The *observation edit distance* is exactly computed if P'_Λ is optimally solved (according to the number of edit actions); is approximated if P'_Λ is solved with a satisfying planner (our case); and is a less accurate estimate (but faster to be computed) if the solved task is a relaxation of P'_Λ [Bonet and Geffner, 2001].

5 Experimental Evaluation

This section evaluates the performance of our approach for learning STRIPS action models from state observations.

Reproducibility

We used 15 IPC domains that satisfy the STRIPS requirement [Fox and Long, 2003], taken from the PLANNING.DOMAINS repository [Muise, 2016]. The training set comprises twenty five states per domain, i.e. $\langle s_0, s_1, \ldots, s_{24} \rangle$, and is fixed for all the experiments so the results reported by different evaluation approaches are comparable. All experiments are run on an Intel Core i5 3.10 GHz x 4 with 8 GB of RAM and a 600 seconds time bound.

	$\delta(\mathcal{M}, \mathcal{O})$	$\delta(\mathcal{M}, *)$	$1 - \frac{\delta(\mathcal{M}, \mathcal{O})}{\delta(\mathcal{M}, *)}$
blocks	0	90	1.0
driverlog	5	144	0.97
ferry	2	69	0.97
floortile	34	342	0.90
grid	42	153	0.73
gripper	2	30	0.93
hanoi	1	63	0.98
hiking	69	174	0.60
miconic	3	72	0.96
npuzzle	2	24	0.92
parking	4	111	0.96
satellite	24	75	0.68
transport	4	78	0.95
visitall	2	24	0.92
zenotravel	3	63	0.95

Table 1: Evaluation of the quality of the learned models with respect to an observations test set.

MADAGASCAR is the classical planner we used to solve the instances that result from our compilations for its ability to deal with dead-ends [Rintanen, 2014]. Due to its SAT-based nature, MADAGASCAR can apply the actions for programming preconditions in a single planning step (in parallel) because there is no interaction between these actions. Actions for programming effects can also be applied in a single planning step, thus significantly reducing the planning horizon.

The compilation source code, evaluation scripts and benchmarks (including the used training and test sets) are fully available at this anonymous repository *https://github.com/anonsub/observations-learning* so any experimental data reported in the paper can be reproduced.

Evaluating with a test set

When a reference model is not available, the learned models are tested with an observation set. Table 1 summarizes the results obtained when evaluating the quality of the learned models with respect to a test set of state observations. Each test set comprises between 20 and 50 observations per domain and is generated executing the plans for various instances of the IPC domains and collecting the intermediate states.

The table shows, for each domain, the *observation edit distance* (computed with our extended compilation), the *maximum edit distance*, and their ratio. The reported results show that, despite learning only from 25 state observations, 12 out of 15 learned domains yield ratios of 90% or above. This fact evidences that the learned models require very small amounts of edition to match the observations of the given test set.

Evaluating with a reference model

Here we evaluate the learned models with respect to the actual generative model. Opposite to what usually happens in ML, this model is available when learning is applied to IPC domains. The model learned for each domain is compared

with its reference model using:

- $Precision = \frac{tp}{tp+fp}$, where tp is the number of *true positives* (predicates that correctly appear in the action model) and fp is the number of *false positives* (predicates of the learned model that should not appear). Precision gives a notion of *soundness*.
- $Recall = \frac{tp}{tp+fn}$, where fn is the number of *false negatives* (predicates that should appear in the learned model but are missing). Recall gives a notion of *completeness*.

Table 2 shows the precision (**P**) and recall (**R**) computed separately for the preconditions (**Pre**), positive effects (**Add**) and negative Effects (**Del**) while the last two columns report averages values. The reason why the scores in Table 2 are lower than in Table 1 is because the syntax-based nature of *precision* and *recall* make these two metrics report low scores for learned models that are semantically correct but correspond to *reformulations* of the actual model (changes in the roles of actions with matching headers or parameters with matching types).

Precision and recall robust to model reformulations

To give an insight of the actual quality of the learned models, we defined a method for computing *Precision* and *Recall* that is robust to the mentioned model *reformulations*.

Precision and recall are often combined using the *harmonic mean*. This expression, called the *F-measure* or the balanced *F-score*, is defined as $F = 2 \times \frac{Precision \times Recall}{Precision + Recall}$. Given the learned action model $\mathcal{M}$ and the reference action model $\mathcal{M}^*$, the bijective function $f_{P\&R} : \mathcal{M} \mapsto \mathcal{M}^*$ is the mapping between the learned and the reference model that maximizes the accumulated *F-measure* (considering swaps in the actions with matching headers or parameters with matching types). Table 3 shows that significantly higher values of *precision* and *recall* are reported when a learned action schema, $\xi \in \mathcal{M}$, is compared to its corresponding reference schema given by the $f_{P\&R}$ mapping ($f_{P\&R}(\xi) \in \mathcal{M}^*$). These results evidence that in all of the evaluated domains, except for *ferry* and *satellite*, the learning task swaps the roles of some actions (or parameters) with respect to their role in the reference model.

As we can see in Table 3, the *blocksworld* and *gripper* domains are perfectly learned from only 25 state observations. On the other hand, the learning scores of several domains in Table 3 are still lower than in Table 1. The reason lies in the particular observations comprised by the test sets. As an example, in the *driverlog* domain, the action schema `disembark-truck` is missing from the learned model because this action is never induced from the observations in the training set; that is, such action never appears in the corresponding *unobserved* plan. The same happens with the `paint-down` action of the *floortile* domain or `move-curb-to-curb` in the *parking* domain. Interestingly, these actions do not appear either in the test sets and so the learned action models are not penalized in Table 1. Generating *informative* and *representative* observations for learning planning action models is an open issue. Planning actions include preconditions that are only satisfied by specific sequences of actions, often, with a low probability of being chosen by chance [Fern *et al.*, 2004].

	Pre		Add		Del			
	P	**R**	**P**	**R**	**P**	**R**	**P**	**R**
blocks	0.44	0.44	0.44	0.44	0.44	0.44	0.44	0.44
driverlog	0.0	0.0	0.25	0.43	0.0	0.0	0.08	0.14
ferry	1.0	0.71	1.0	1.0	1.0	1.0	1.0	0.9
floortile	0.38	0.55	0.4	0.18	0.56	0.45	0.44	0.39
grid	0.5	0.47	0.33	0.29	0.25	0.29	0.36	0.35
gripper	0.83	0.83	0.75	0.75	0.75	0.75	0.78	0.78
hanoi	0.5	0.25	0.5	0.5	0.0	0.0	0.33	0.25
hiking	0.43	0.43	0.5	0.35	0.44	0.47	0.46	0.42
miconic	0.6	0.33	0.33	0.25	0.33	0.33	0.42	0.31
npuzzle	0.33	0.33	0.0	0.0	0.0	0.0	0.11	0.11
parking	0.25	0.21	0.0	0.0	0.0	0.0	0.08	0.07
satellite	0.6	0.21	0.8	0.8	1.0	0.5	0.8	0.5
transport	1.0	0.3	0.8	0.8	1.0	0.6	0.93	0.57
visitall	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
zenotravel	0.67	0.29	0.33	0.29	0.33	0.14	0.44	0.24

Table 2: Precision and recall values obtained without computing the $f_{P\&R}$ mapping with the reference model.

	Pre		Add		Del			
	P	**R**	**P**	**R**	**P**	**R**	**P**	**R**
blocks	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
driverlog	0.67	0.14	0.33	0.57	0.67	0.29	0.56	0.33
ferry	1.0	0.71	1.0	1.0	1.0	1.0	1.0	0.9
floortile	0.44	0.64	1.0	0.45	0.89	0.73	0.78	0.61
grid	0.63	0.59	0.67	0.57	0.63	0.71	0.64	0.62
gripper	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
hanoi	1.0	0.5	1.0	1.0	1.0	1.0	1.0	0.83
hiking	0.78	0.6	0.93	0.82	0.88	0.88	0.87	0.77
miconic	0.8	0.44	1.0	0.75	1.0	1.0	0.93	0.73
npuzzle	0.67	0.67	1.0	1.0	1.0	1.0	0.89	0.89
parking	0.56	0.36	0.5	0.33	0.5	0.33	0.52	0.34
satellite	0.6	0.21	0.8	0.8	1.0	0.5	0.8	0.5
transport	1.0	0.3	1.0	1.0	1.0	0.6	1.0	0.63
visitall	0.67	1.0	1.0	1.0	1.0	1.0	0.89	1.0
zenotravel	1.0	0.43	0.67	0.57	1.0	0.43	0.89	0.48

Table 3: Precision and recall values obtained when computing the $f_{P\&R}$ mapping with the reference model.

6 Conclusions

Unlike extensive-data ML approaches, our work explores an alternative research direction to learn sound models from small amounts of state observations. To the best of our knowledge, this is the first work on learning STRIPS action models from state observations, using exclusively planning technology, and evaluated over a wide range of different domains. Recently, the work in [Stern and Juba, 2017] proposes a planning compilation for learning action models from plan traces following the *finite domain* representation for the state variables. This is a theoretical study on the boundaries of the learned models and no experimental results are reported.

Unlike statistical learning, our inference-based approach is able to produce good-quality models from very small data sets. The action models of the *blocksworld* or *gripper* domains were perfectly learned from only 25 state observations. Moreover, in 12 out of the 15 domains, the learned models yield *Precision* values over 0.75. The success of recent algorithms for exploring planning tasks [Francès *et al.*, 2017] motivates the development of novel techniques to autonomously collect informative learning examples. The combination of such exploration techniques with our learning approach is an appealing research direction towards the bootstrapping of planning action models.

References

[Amir and Chang, 2008] Eyal Amir and Allen Chang. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33:349–402, 2008.

[Asai and Fugunaga, 2018] Masataro Asai and Alex Fugunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *National Conference on Artificial Intelligence, AAAI-18*, 2018.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Bonet *et al.*, 2009] Blai Bonet, Héctor Palacios, and Héctor Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

[Cresswell *et al.*, 2013] Stephen N Cresswell, Thomas L McCluskey, and Margaret M West. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(02):195–213, 2013.

[Fern *et al.*, 2004] Alan Fern, Sung Wook Yoon, and Robert Givan. Learning domain-specific control knowledge from random walks. In *International Conference on Automated Planning and Scheduling, ICAPS-04*, pages 191–199, 2004.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Francès *et al.*, 2017] Guillem Francès, Miquel Ramírez, Nir Lipovetzky, and Hector Geffner. Purely declarative action descriptions are overrated: Classical planning with simulators. In *International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4294–4301, 2017.

[Gregory and Cresswell, 2016] Peter Gregory and Stephen Cresswell. Domain model acquisition in the presence of static relations in the LOP system. In *International Joint Conference on Artificial Intelligence, IJCAI-16*, pages 4160–4164, 2016.

[Kambhampati, 2007] Subbarao Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *National Conference on Artificial Intelligence, AAAI-07*, 2007.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language, 1998.

[Muise, 2016] Christian Muise. Planning. domains. *ICAPS system demonstration*, 2016.

[Ramírez and Geffner, 2009] Miquel Ramírez and Hector Geffner. Plan recognition as planning. In *International Joint conference on Artifical Intelligence*, pages 1778–1783, 2009.

[Rintanen, 2014] Jussi Rintanen. Madagascar: Scalable planning with sat. *Proceedings of the 8th International Planning Competition (IPC-2014)*, 2014.

[Segovia-Aguas *et al.*, 2016] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Hierarchical finite state controllers for generalized planning. In *International Joint Conference on Artificial Intelligence, IJCAI-16*, pages 3235–3241. AAAI Press, 2016.

[Segovia-Aguas *et al.*, 2017] Javier Segovia-Aguas, Sergio Jiménez, and Anders Jonsson. Generating context-free grammars using classical planning. In *International Joint Conference on Artificial Intelligence, ICAPS-17*, 2017.

[Slaney and Thiébaux, 2001] John Slaney and Sylvie Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.

[Sohrabi *et al.*, 2016] Shirin Sohrabi, Anton V. Riabov, and Octavian Udrea. Plan recognition as planning revisited. In *International Joint Conference on Artificial Intelligence, IJCAI-16*, pages 3258–3264, 2016.

[Stern and Juba, 2017] Roni Stern and Brendan Juba. Efficient, safe, and probably approximately complete learning of action models. In *International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4405–4411, 2017.

[Yang *et al.*, 2007] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143, 2007.

[Zhuo and Kambhampati, 2013] Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *International Joint Conference on Artificial Intelligence, IJCAI-13*, pages 2444–2450, 2013.