One-shot learning: From domain knowledge to action models

Abstract

Most approaches to learning action planning models heavily rely on a significantly large volume of training samples or plan observations. In this paper, we adopt a different approach based on deductive learning from domain-specific knowledge, specifically from logic formulae that specify constraints about the possible states of a given domain. The minimal input observability required by our approach is a single example composed of a full initial state and a partial goal state. We will show that exploiting specific domain knowledge enable to constrain the space of possible action models as well as to complete partial observations, both of which turn out helpful to learn good-quality action models.

1 Introduction

The learning of action models in planning has been typically addressed with inductive learning data-intensive approaches. From the pioneer learning system ARMS [Yang *et al.*, 2007] to more recent ones [Mourão *et al.*, 2012; Zhuo and Kambhampati, 2013; Kucera and Barták, 2018], all of them require thousands of plan observations or training samples, i.e., sequences of actions as evidence of the execution of an observed agent, to obtain and validate an action model. These approaches return the statistically significant model that best explains the plan observations by minimizing some error metric. A model explains an observation if a plan containing the observed actions is computable with the model and the states induced by this plan also include the possibly partially observed states. The limitation of posing model validation as an optimization task over a testing set of observations is that it neither guarantees completeness (the model may not explain all the observations) nor correctness (the states induced by the execution of the plan generated with the model may contain contradictory information).

Differently, other approaches rely on symbolic-via learning. The Simultaneous Learning and Filtering (SLAF) approach [Amir and Chang, 2008] exploits logical inference and builds a complete explanation through a CNF formula that represents the initial belief state, and a plan observation. The formula is updated with every action and state of the observation, thus representing all possible transition relations consistent with it. SLAF extracts all satisfying models of the learned formula with a SAT solver although the algorithm cannot effectively learn the preconditions of actions. A more recent approach addresses the learning of action models from plan observations as a planning task which searches the space of all possible action models [Aineto *et al.*, 2018]. A plan here is conceived as a series of steps that determine the preconditions and effects of the action models plus other steps that validate the formed actions in the observations. The advantage of this approach is that it only requires input samples of about a total of 50 actions.

This paper studies the impact of using mixed input data, i.e, automatically-collected plan observations and human-encoded domain-specific knowledge, in the learning of action models. Particularly, we aim to stress the extreme case of having a single observation sample and answer the question to whether the lack of training samples can be overcome with the supply of domain knowledge. The question is motivated by (a) the assumption that obtaining enough training observations is often difficult and costly, if not impossible in some domains [Zhuo, 2015]; (b) the fact that although the physics of the real-world domain being modeled are unknown, the user may know certain pieces of knowledge about the domain; and (c) the desire for correct action models that are usable beyond their applicability to a set of testing observations. To this end, we opted for checking our hypothesis in the framework proposed in [Aineto *et al.*, 2018] since this planning-based satisfiability approach allows us to configure additional constraints in the compilation scheme, it is able to work under a minimal set of observations and uses an off-the-shelf planner[1]. Ultimately, we aim to compare the informational power of domain observations (information quantity) with the representational power of domain-specific knowledge (information quality). Complementarily, we restrict our attention to solely observations over fluents as in many applications the actual actions of an agent may not be observable [Sohrabi *et al.*, 2016].

Next section summarizes basic planning concepts and outlines the baseline learning approach [Aineto *et al.*, 2018].

[1]We thank authors for providing us with the source files of their learning system.

Then we formalize our one-shot learning task with domain knowledge and subsequently we explain the task-solving process. Section 5 presents the experimental evaluation and last section concludes.

2 Background

We denote as F (fluents) the set of propositional state variables. A partial assignment of values to fluents is represented by L (literals). We adopt the *open world assumption* (what is not known to be true in a state is unknown) to implicitly represent the unobserved literals of a state. Hence, a state s includes positive literals (f) and negative literals ($\neg f$) and it is defined as a full assignment of values to fluents; $|s| = |F|$. 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 *planning action* a has a precondition list $\mathsf{pre}(a) \in \mathcal{L}(F)$ and a effect list $\mathsf{eff}(a) \in \mathcal{L}(F)$. The semantics of an action a is specified with two functions: $\rho(s, a)$ denotes whether a is *applicable* in a state s and $\theta(s, a)$ denotes the *successor state* that results from applying a in a state s. Then, $\rho(s, a)$ holds iff $\mathsf{pre}(a) \subseteq s$, i.e. if its preconditions hold in s. The result of executing an applicable action a in a state s is a new state $\theta(s, a) = \{s \setminus \neg\mathsf{eff}(a) \cup \mathsf{eff}(a)\}$, where $\neg\mathsf{eff}(a)$ is the complement of $\mathsf{eff}(a)$, which is subtracted from s so as to ensure that $\theta(s, a)$ remains a well-defined state. The subset of effects of an action a that assign a positive value to a fluent is called *positive effects* and denoted by $\mathsf{eff}^+(a) \in \mathsf{eff}(a)$ while $\mathsf{eff}^-(a) \in \mathsf{eff}(a)$ denotes the *negative effects*.

A *planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where I is the initial state and $G \in \mathcal{L}(F)$ is the set of goal conditions over the state variables. A *plan* π is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$, with $|\pi| = n$ denoting its *plan length*. The execution of π in I induces a *trajectory* $\langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, it holds $\rho(s_{i-1}, a_i)$ and $s_i = \theta(s_{i-1}, a_i)$. A plan π solves P iff the induced trajectory reaches a final state s_n such that $G \subseteq s_n$.

The baseline learning approach our proposal draws upon uses *actions with conditional effects* [Aineto *et al.*, 2018]. The conditional effects of an action a_c is composed of two sets of literals: $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*. The *triggered effects* resulting from the action application (conditional effects whose conditions hold in s) is defined as $\mathsf{eff}_c(s, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a_c), C \subseteq s} E$.

2.1 Learning action models as planning

The approach for learning STRIPS action models presented in [Aineto *et al.*, 2018], which we will use as our baseline learning system (hereafter BLS, for short), is a compilation scheme that transforms the problem of learning the preconditions and effects of action models into a planning task P'. A STRIPS *action model* ξ is defined as $\xi = \langle name(\xi), pars(\xi), pre(\xi), add(\xi), del(\xi)\rangle$, where $name(\xi)$ and parameters, $pars(\xi)$, define the header of ξ; and $pre(\xi)$, $del(\xi)$ and $add(\xi)$) are sets of fluents that represent the *preconditions*, *negative effects* and *positive effects*, respectively, of the actions induced from the action model ξ.

The BLS receives as input an empty domain model, which only contains the headers of the action models, and a set of observations of plan executions, and creates a propositional encoding of the planning task P'. Let Ψ be the set of *predicates*[2] that shape the variables F. The set of propositions of P' that can appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of a given ξ, denoted as $\mathcal{I}_{\Psi,\xi}$, are FOL interpretations of Ψ over the parameters $pars(\xi)$. For instance, in a four-operator *blocksworld* [Slaney and Thiébaux, 2001], the $\mathcal{I}_{\Psi,\xi}$ set contains five elements for the $\texttt{pickup}(v_1)$ model, $\mathcal{I}_{\Psi,pickup}$={$\texttt{handempty}$, $\texttt{holding}(v_1)$, $\texttt{clear}(v_1)$, $\texttt{ontable}(v_1)$, $\texttt{on}(v_1, v_1)$} and eleven elements for the model of $\texttt{stack}(v_1, v_2)$, $\mathcal{I}_{\Psi,stack}$={$\texttt{handempty}$, $\texttt{holding}(v_1)$, $\texttt{holding}(v_2)$, $\texttt{clear}(v_1)$, $\texttt{clear}(v_2)$, $\texttt{ontable}(v_1)$, $\texttt{ontable}(v_2)$, $\texttt{on}(v_1, v_1)$, $\texttt{on}(v_1, v_2)$, $\texttt{on}(v_2, v_1)$, $\texttt{on}(v_2, v_2)$}. Hence, solving P' consists in determining which elements of $\mathcal{I}_{\Psi,\xi}$ will shape the preconditions, positive and negative effects of the action model ξ.

The decision as to whether or not an element of $\mathcal{I}_{\Psi,\xi}$ will be part of $pre(\xi)$, $del(\xi)$ or $add(\xi)$ is given by the plan that solves P'. Specifically, two different sets of actions are included in the definition of P': *insert actions*, which insert preconditions and effects on an action model; and *apply actions*, which validate the application of the learned action models in the input observations. Roughly speaking, in the *blocksworld* domain, the *insert actions* of a plan that solves P' will look like `(insert_pre_stack_holding_v1)`, `(insert_eff_stack_clear_v1)`, `(insert_eff_stack_clear_v2)`, where the second action denotes a positive effect and the third one a negative effect both to be inserted in the model of `stack`; and the second set of actions of the plan that solves P' will be like `(apply_unstack blockB blockA)`, `(apply_putdown blockB)` and `(validate_1)`, `(validate_2)`, where the last two actions denote the points at which the states generated through the `apply` actions must be validated with the observations of plan executions.

In a nutshell, the output of the BLS compilation is a plan that completes the empty input domain model by specifying the preconditions and effects of each action model such that the validation of the completed model over the input observations is successful.

3 *One-shot* learning task

The *one-shot* learning task to learn action models from *domain-specific knowledge* is defined as a tuple $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi\rangle$, where:

- $\mathcal{M}$ is the *initial empty model* that contains only the header of each action model to be learned.
- $\mathcal{O}$ is a single learning example or plan observation; i.e. a sequence of (partially) observable states representing the evidence of the execution of an observed agent the observation of a sequence of states generated with the aimed planning action model?????.

[2]The initial state of an observation is a full assignment of values to fluents, $|s_0| = |F|$, and so the predicates Ψ are extractable from the observed state $|s_0|$.

- Φ is a set of logic formulae that define *domain-specific knowledge*.

A *solution* to a learning task $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ is a model $\mathcal{M}'$ s.t. there exists a plan computable with $\mathcal{M}'$ that is consistent with the headers of $\mathcal{M}$, the observed states of $\mathcal{O}$ and the given domain knowledge in Φ.

3.1 The space of STRIPS action models

We analyze here the search space of a learning task $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$; i.e., the space of STRIPS action models. In principle, for a given action model ξ, any element of $\mathcal{I}_{\Psi,\xi}$ can potentially appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$. In practice, the actual space of possible STRIPS schemata is bounded by:

1. **Syntactic constraints**. The solution $\mathcal{M}'$ must be consistent with the STRIPS constraints: $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$.
2. **Observation constraints**. The solution $\mathcal{M}'$ must be consistent with these *semantic constraints* derived from learning samples, which in our case is a single plan observation. Specifically, the states induced by the plan computable with M' must comprise the observed states of the sample.

Considering only the syntactic constraints, the size of the space of possible STRIPS schemata is given by $2^{2\times|\mathcal{I}_{\Psi,\xi}|}$ because one element in $\mathcal{I}_{\Psi,\xi}$ can appear both in the preconditions and effects of ξ. *Typing constraints* would also be a type of syntactic constraint [McDermott *et al.*, 1998] in non-STRIPS models. Additionally, observation samples further constrains the space of possible action models.

==================================

HABLAR DE ESTO

In this work we introduce a novel propositional encoding of the *preconditions*, *negative*, and *positive* effects of a STRIPS action schema ξ that uses only fluents of two kinds $\texttt{pre\_e\_}\xi$ and $\texttt{eff\_e\_}\xi$ (where $e \in \mathcal{I}_{\Psi,\xi}$). This encoding exploits the syntactic constraints of STRIPS so it is more compact that the one previously proposed by Aineto *et al.* 2018 for learning STRIPS action models with classical planning. In more detail, if $\texttt{pre\_e\_}\xi$ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *precondition* in ξ. If $\texttt{pre\_e\_}\xi$ and $\texttt{eff\_e\_}\xi$ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *negative effect* in ξ while if pre_e_ξ does not hold but $\texttt{eff\_e\_}\xi$ holds, it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *positive effect* in ξ. Figure 1 shows the PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema using the `pre_e_stack` and `eff_e_stack` fluents ($e \in \mathcal{I}_{\Psi,stack}$).

===

3.2 The sampling space

The single plan observation is defined as $\mathcal{O} = \langle s_0^o, s_1^o \ldots, s_m^o \rangle$, a sequence of possibly *partially observed states* except for the initial state s_0^o which is a *fully observable* state. The set of predicates Ψ and the set of objects Ω that shape the fluents F is then deducible from $\mathcal{O}$. A partially observed state s_i^o, $1 \leq i \leq m$, is one in which $|s_i^o| < |F|$; i.e., a state in which at least a fluent of F was not observed. Intermediate states can be *missing*, meaning that they are *unobserved*, so transiting between two consecutive observed states in $\mathcal{O}$ may require the execution of more than a single action ($\theta(s_i^o, \langle a_1, \ldots, a_k \rangle) = s_{i+1}^o$ (where $k \geq 1$ is unknown but finite). The minimal expression of a learning example must comprise at least two state observations, a full initial state s_0^o and a partially observed state s_m^o so $m \geq 1$.

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


(pre_holding_v1_stack) (pre_clear_v2_stack)
(eff_holding_v1_stack) (eff_clear_v2_stack)
(eff_clear_v1_stack) (eff_handempty_stack) (eff_on_v1_v2_stack)
```

Figure 1: PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema.

Figure 2 shows a learning example that contains an initial state of the blocksworld where the robot hand is empty and three blocks (namely `blockA`, `blockB` and `blockC`) are on top of the table and clear. The observation represents a partially observed state in which `blockA` is on top of `blockB` and `blockB` on top of `blockC`.

```
(:predicates (on ?x ?y) (ontable ?x)
     (clear ?x) (handempty)
     (holding ?x))

(:objects blockA blockB blockC)

(:init (ontable blockA) (clear blockA)
       (ontable blockB) (clear blockB)
       (ontable blockC) (clear blockC)
       (handempty))

(:observation (on blockA blockB) (on blockB blockC))
```

Figure 2: Example of a two-state observationn for the learning STRIPS action models.

3.3 The domain-specific knowledge

Our approach is to introduce *domain-specific knowledge* in the form of *state-constraints* to restrict further the space of possible schemata. For instance, in the *blocksworld* one can argue that $\texttt{on}(v_1, v_1)$ and $\texttt{on}(v_2, v_2)$ will not appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ lists of an action schema ξ because, in this specific domain, a block cannot be on top of itself. The notion of *state-constraint* is very general and has been used in different areas of AI and for different purposes. If we restrict ourselves to planning, *state-constraints* are abstractions for the compact specification of a set of states. For example, *state-constraints* in planning allow to specify the set of states where a given action is applicable, the set of states where a given *axiom* or *derived predicate* holds or the set of states that are considered goal states.

State-invariants is a kind of state-constraints useful for computing more compact state representations of a given planning problem [Helmert, 2009] and for making *satisfiability planning* or *backward search* more efficient [Rintanen, 2014; Alcázar and Torralba, 2015]. Given a classical planning problem $P = \langle F, A, I, G \rangle$, a *state-invariant* is a formula

ϕ that holds at the initial state of a given classical planning problem, $I \models \phi$, and at every state s, built from F, that is reachable from I by applying actions in A. For instance, Figure 3 shows five clauses that define *state-invariants* for the *blocksworld* planning domain.

$\forall x_1, x_2\ ontable(x_1) \leftrightarrow \neg on(x_1, x_2)$.
$\forall x_1, x_2\ clear(x_1) \leftrightarrow \neg on(x_2, x_1)$.
$\forall x_1, x_2, x_3\ \neg on(x_1, x_2) \vee \neg on(x_1, x_3)\ such\ that\ x_2 \neq x_3$.
$\forall x_1, x_2, x_3\ \neg on(x_2, x_1) \vee \neg on(x_3, x_1)\ such\ that\ x_2 \neq x_3$.
$\forall x_1, \ldots, x_n\ \neg(on(x_1, x_2) \wedge on(x_2, x_3) \wedge \ldots \wedge on(x_{n-1}, x_n) \wedge on(x_n, x_1))$.

Figure 3: Example of *state-invariants* for the *blocksworld* domain.

A *mutex* (mutually exclusive) is a *state-invariant* that takes the form of a binary clause and indicates a pair of different properties that cannot be simultaneously true [Kautz and Selman, 1999]. For instance in a three-block *blocksworld*, $\neg on(block_A, block_B) \vee \neg on(block_A, block_C)$ is a *mutex* because $block_A$ can only be on top of a single block.

A *domain invariant* is an instance-independent state-invariant, i.e. holds for any possible initial state and any possible set of objects. Therefore, if a given state s holds $s \not\models \phi$ such that ϕ is a *domain invariant*, it means that s is not a valid state. Domain invariants are often compactly defined as *lifted invariants* (also called *schematic* invariants [Rintanen, 2017]).

In this work we exploit *domain-specific knowledge* that is given as *schematic mutex*. We pay special attention to *schematic mutex* because they identify mutually exclusive *properties* of a given type of objects [Fox and Long, 1998] and because they enable (1) effective completion of partially observed states and (2) effectively pruning of inconsistent STRIPS action models. We define a *schematic mutex* as a $\langle p, q\rangle$ pair where both $p, q \in \mathcal{I}_{\Psi,\xi}$ are predicates that shape the preconditions or effects of a given action scheme ξ and such that they satisfy the formulae $\neg p \vee \neg q$, considering that their corresponding variables are *universally quantified*. For instance, $holding(v_1)$ and $clear(v_1)$ from the *blocksworld* are *schematic mutex* while $clear(v_1)$ and $ontable(v_1)$ are not because $\forall v_1, \neg clear(v_1) \vee \neg ontable(v_1)$ does not hold for every possible *blocksworld* state.

4 Action model learning from *schematic mutexes*

This section shows how to solve the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi\rangle$ learning task with an off-the-shelf classical planner.

4.1 Completing partially observed states with *schematic mutexes*

Here we describe a pre-processing mechanism to add new literals to complete the partial states $\langle s_1^o \ldots, s_m^o\rangle$ of an observation using a set of *schematic mutexes* Φ.

Let us define the production rule $p \rightarrow \neg q$ such that $\langle p, q\rangle$ is a *schematic mutex*. Given a *schematic mutex* $\langle p, q\rangle \in \Phi$ and a state observation $s_j^o \in \mathcal{O}$, $(1 \leq j \leq m)$ then s_j^o can be safely completed adding the new literals $\neg q(\omega)$ that result from the unification of the corresponding production rule with s_j^o. $\omega \subseteq \Omega^{pars(q)}$ represents the objects that unify the variables in q such that Ω^k is the k-th Cartesian power of Ω. For instance, if the literal `holding(blockA)` is observed in a particular blocksword state and Φ contains the *schematic mutex* $\neg holding(v_1) \vee \neg clear(v_1)$, we can safely extend that state observation with literal `¬clear(blockA)` (despite this particular literal being initially unknown).

ID	Action	New conditional effect
1	$\mathsf{insertPre}_{\mathsf{p},\xi}$	$\{pre\_q\_\xi\} \rhd \{invalid\}$
2	$\mathsf{insertEff}_{\mathsf{p},\xi}$	$\{pre\_q\_\xi \wedge eff\_q\_\xi \wedge pre\_p\_\xi\} \rhd \{invalid\}$
3	$\mathsf{insertEff}_{\mathsf{p},\xi}$	$\{\neg pre\_q\_\xi \wedge eff\_q\_\xi \wedge \neg pre\_p\_\xi\} \rhd \{invalid\}$
4	$\mathsf{apply}_{\xi,\omega}$	$\{\neg pre\_p\_\xi \wedge eff\_p\_\xi \wedge$ $q(\omega) \wedge \neg pre\_q\_\xi\} \rhd \{invalid\}$
5	$\mathsf{apply}_{\xi,\omega}$	$\{\neg pre\_p\_\xi \wedge eff\_p\_\xi \wedge$ $q(\omega) \wedge pre\_q\_\xi \wedge \neg eff\_q\_\xi\} \rhd \{invalid\}$

Figure 4: Summary of the new conditional effects added to the classical planning compilation for the learning of STRIPS action models.

4.2 Pruning inconsistent action models with *schematic mutexes*

Our approach to learning action models consistent with the set of *state-constraints* in Φ is to ensure that newly generated states produced by the learned actions cannot introduce any inconsistencies. This is implemented by adding new conditional effects to the *insert* and *apply* actions of the classical planning compilation. Figure 4 summarizes the new conditional effects added to the compilation and next, we describe them in detail:

1-3 For every *schematic mutex* $\langle p, q\rangle$ s.t. both p and q belong to $\in \mathcal{I}_{\Psi,\xi}$ one conditional effect is added to the $\mathsf{insertPre}_{\mathsf{p},\xi}$ actions to ban the insertion of two preconditions that are *schematic mutex*. Likewise, two conditional effects are added to the $\mathsf{insertEff}_{\mathsf{p},\xi}$ actions, one to ban the insertion of two positive effects that are *schematic mutex* and another one to ban two mutex negative effects.

4-5 For every *schematic mutex* $\langle p, q\rangle$ s.t. both p and q belong to $\in \mathcal{I}_{\Psi,\xi}$ two conditional effects are added to the $\mathsf{apply}_{\xi,\omega}$ actions to ban positive effects that are inconsistent with an input observation (in $\mathsf{apply}_{\xi,\omega}$ actions the variables in $pars(\xi)$ are bounded to the objects in ω that appear in the same position).

In theory, conditional effects of the kind 4 and 5 are enough to guarantee that all the states traversed by a plan produced by the compilation are *consistent* with the input set of *schematic mutexes* Φ (of course, provided that the input initial state s_0^o is a valid state). In practice we include also conditional effects of the kind 1, 2 and 3 because they prune *invalid* action models at an earlier stage of the planning process (these effects extend the *insert* actions that always appear first in the solution plans).

The goals of the classical planning problem output by the original compilation are extended with the $\neg invalid$ literal to validate that only states *consistent* with the state constraints defined in Φ are traversed by solution plans. Remarkably, the $\neg invalid$ literal allows us also to define $\mathsf{apply}_{\xi,\omega}$ actions more compactly than in the original compilation by Aineto *et al.* 2018. Disjunctions are no longer required to code the possible preconditions of an action schema since they can now be encoded with conditional effects of this kind $\{pre\_p\_\xi \wedge \neg p(\omega)\} \rhd \{invalid\}$.

4.3 Compilation properties

Lemma 1. *Soundness. Any classical plan π_Λ that solves P_Λ produces a* STRIPS *model $\mathcal{M}'$ that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task.*

Proof. According to the P_Λ compilation, once a given precondition or effect is inserted into the domain model $\mathcal{M}$ it cannot be removed back. In addition, once an action model is applied it cannot be modified. In the compiled planning problem P_Λ, only $\mathsf{apply}_{\xi,\omega}$ actions can update the value of the state fluents F. This means that a state consistent with an observation s_n^o can only be achieved executing an applicable sequence of $\mathsf{apply}_{\xi,\omega}$ actions that, starting in the corresponding initial state s_0^o, validates that every generated intermediate state s_i, s.t. $0 \leq i \leq n$, is consistent with the input state observations and *state-invariants*. This is exactly the definition of the solution condition for model $\mathcal{M}'$ to solve the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task. □

Lemma 2. *Completeness. Any* STRIPS *model $\mathcal{M}'$ that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task can be computed with a classical plan π_Λ that solves P_Λ.*

Proof. By definition $\mathcal{I}_{\Psi,\xi}$ fully captures the set of elements that can appear in an action model ξ using predicates Ψ. In addition the P_Λ compilation does not discard any possible domain model $\mathcal{M}'$ definable within $\mathcal{I}_{\Psi,\xi}$ that satisfies the mutexes in Φ. This means that, for every model $\mathcal{M}'$ that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$, we can build a plan π_Λ that solves P_Λ by selecting the appropriate $\mathsf{insertPre}_{\mathsf{p},\xi}$ and $\mathsf{insertEff}_{\mathsf{p},\xi}$ actions for *programming* the precondition and effects of the corresponding action models in $\mathcal{M}'$ and then, selecting the corresponding $\mathsf{apply}_{\xi,\omega}$ actions that transform the initial state observation s_0^o into the final state observation s_m^o. □

The size of the classical planning task P_Λ output by our compilation depends on the arity of the given *predicates* Ψ, that shape the propositional state variables F, and the number of parameters of the action models, $|pars(\xi)|$. The larger these arities, the larger $|\mathcal{I}_{\Psi,\xi}|$. The size of the $\mathcal{I}_{\Psi,\xi}$ set is the term that dominates the compilation size because it defines the pre_e_ξ/eff_e_ξ fluents, the corresponding set of *insert* actions, and the number of conditional effects in the $\mathsf{apply}_{\xi,\omega}$ actions. Note that *typing* can be used straightforward to constrain the FOL interpretations of Ψ over the parameters $pars(\xi)$ which significantly reduces $|\mathcal{I}_{\Psi,\xi}|$ and hence, the size of the classical planning task output by the compilation.

Classical planners tend to preffer shorter solution plans, so our compilation may introduce a bias to $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning tasks preferring solutions that are referred to action models with a shorter number of *preconditions/effects*. In more detail, all $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ fluents are false at the initial state of our P_Λ compilation so classical planners tend to solve P_Λ with plans that require a smaller number of *insert* actions.

This bias could be eliminated defining a cost function for the actions in P_Λ (e.g. *insert* actions have *zero cost* while $\mathsf{apply}_{\xi,\omega}$ actions have a *positive constant cost*). In practice we use a different approach to disregard the cost of *insert* actions since classical planners are not proficient at optimizing *plan cost* when there are zero-cost actions. Instead, our approach is to use a SAT-based planner [Rintanen, 2014] that can apply all actions for inserting preconditions in a single planning step (these actions do not interact). Further, the actions for inserting action effects are also applied in another single planning step. The plan horizon for programming any action model is then always bound to 2, which significantly reduces the planning horizon. The SAT-based planning approach is also convenient because its ability to deal with classical planning problems populated with dead-ends and because symmetries in the insertion of preconditions/effects into an action model do not affect the planning performance.

5 Evaluation

This section evaluates the performance of our approach for learning STRIPS action models with different amounts of available input knowledge.

Reproducibility

The domains used in the evaluation are IPC domains that satisfy the STRIPS requirement [Fox and Long, 2003], taken from the PLANNING.DOMAINS repository [Muise, 2016]. For each domain we generated 10 trajectories of length 10 via random walks to be used as training examples through all the experiments. We also introduce a new parameter, the *degree of observability* σ, which indicates de probability of observing a literal in an intermediate state. This parameter is used to build training examples with varying degrees of incompleteness from the generated trajectories. All experiments are run on an Intel Core i5 3.10 GHz x 4 with 16 GB of RAM.

For the sake of reproducibility, the compilation source code, evaluation scripts, used benchmarks and input *state-invariants* are fully available at the repository *https://github.com/anonsub/*.

Metrics

The learned models are evaluated using the *precision* and *recall* metrics for action models proposed in [Aineto *et al.*, 2018], which compare the learned models against the reference model.

Precision measures the correctness of the learned models. Formally, $Precision = \frac{tp}{tp+fp}$, where tp is the number of true positives (predicates that appear in both the learned and reference action models) and fp is the number of false positives (predicates that appear in the learned action model but not in the reference model). Recall, on the other hand, measures the completeness of the model and is formally defined as $Recall = \frac{tp}{tp+fn}$ where fn is the number of false negatives (predicates that should appear in the learned action model but are missing).

5.1 Observability versus Knowledge

In our first experiment we seek to answer the question of whether knowledge can substitute observations. To that end, we evaluate the following 4 settings:

- **Neither observability nor knowledge:** This is the baseline setting where the input sample is reduced to the minimum and only the initial and final states are known ($\sigma = 0$).
- **Only knowledge:** Here we add domain-specific knowledge encoded as schematic mutexes to the baseline scenario.
- **Only observability:** In this one, instead of knowledge we use a more complete input example where part of the intermediate states is known ($\sigma = 0.2$).
- **Both observability and knowledge:** In the last setting we use both more complete input examples and schematic mutexes.

		$\sigma = 0$		$\sigma = 0$ with Φ		$\sigma = 0.2$		$\sigma = 0.2$ with Φ	
	$\lvert\Phi\rvert$	P	R	P	R	P	R	P	R
blocks	9	0.52	0.38	0.53	0.21	0.66	0.56	0.77	0.68
driverlog	8	0.49	0.33	0.33	0.31	0.54	0.38	0.70	0.53
ferry	2	0.50	0.40	0.57	0.41	0.59	0.64	0.59	0.70
floor-tile	7	0.30	0.40	0.58	0.46	0.68	0.46	0.75	0.48
grid	3	0.47	0.40	0.47	0.37	0.43	0.34	0.43	0.32
gripper	5	0.77	0.56	0.77	0.54	0.85	0.74	0.96	0.83
hanoi	3	0.84	0.76	0.76	0.75	0.96	0.75	0.97	0.79
n-puzzle	3	0.94	0.86	0.93	0.86	0.99	0.87	1.00	0.87
parking	8	0.48	0.37	0.60	0.40	0.58	0.45	0.67	0.49
transport	4	0.45	0.45	0.53	0.46	0.99	0.51	0.94	0.79
zeno-travel	4	0.72	0.39	0.75	0.40	0.80	0.42	0.93	0.55
		0.59	0.48	0.62	0.47	0.73	0.56	0.79	0.64

Table 1: Observability versus knowledge

Table 1 compiles the average precision (P) and recall (R) for each domain in the different settings tested. The table also reports the number of schematic mutexes ($|\Phi|$) used for each domain. Comparing the settings wtih only knowledge and only observability, it is clear that having more complete training examples is preferable. In fact, the improvement of using domain knowledge is marginal with respect to the baseline case. This is not the case for the last setting, where the use of domain knowledge shows a significant improvement in the quality of the learned models when compared to the setting with only observability.

5.2 Using knowledge to counter incompleteness

In the previous experiment we have stablished that domain knowledge is no substitute to observations, and that, given a minimum of observability, domain knowledge is able to enrich both the observations and learning process thus procuring better learned models. In this next experiment we are going to measure the improvement provided by domain knowledge at increasing degrees of observability.

Figures 5 and 6 compare the precision and recall of the learned models in the settings with and without domain knowledge. The values plotted in these figures are averages across all the domains seen in the previous experiment. The results show that using domain knowledge significantly improves the learned models no matter how complete the training examples are. Another interesting aspect is that domain knowledge is able to enrich observations in the range of 30% observability to the level of fully observable trajectories, which means that domain knowledge can make up for a lack of completeness in the training examples.

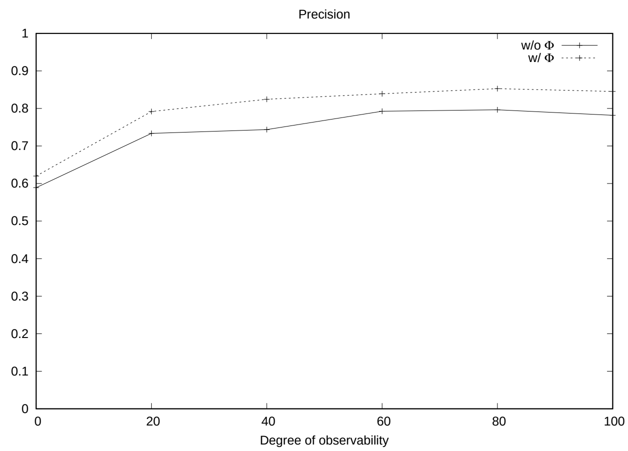

Figure 5: Comparison of the precision of the learned models for increasing degrees of observability.

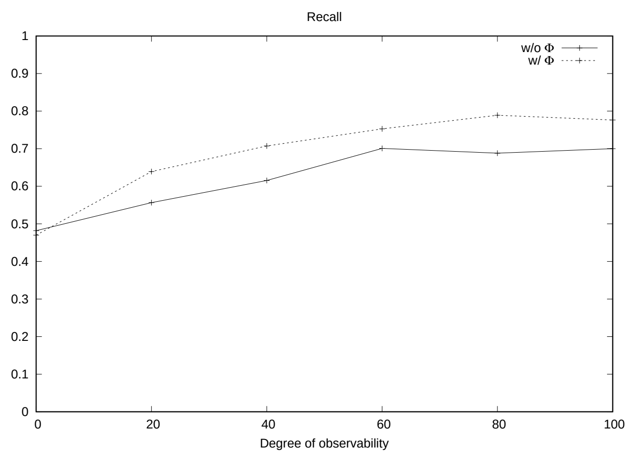

Figure 6: Comparison of the recall of the learned models for increasing degrees of observability.

6 Related work

In *Inductive Logic Programming* it is common to make the hypothesis be consistent with the *background knowledge*, that is some form *deductive knowledge* apart from the examples [Muggleton and De Raedt, 1994].

State-invariants have also been previously used to improve the automatic construction of HTN planning model [Lotinac and Jonsson, 2016].

Our learning setting is related to the classical planning formulation where no action model is given [Stern and Juba, 2017]. This planning setting can can be seen as an scenario when the action model is *learned* from a single example that contains only two state observations: the initial state and the goals.

7 Conclusions

In some contexts it is however reasonable to assume that the action model is not learned from scratch, e.g. because some parts of the action model are known [Zhuo *et al.*, 2013; Sreedharan *et al.*, 2018; Pereira and Meneguzzi, 2018]. Our compilation is also flexible to this particular learning scenario. The known preconditions and effects are encoded setting the corresponding fluents $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ to true in the initial state. Further, the corresponding insert actions, $\mathsf{insertPre}_{\mathsf{p},\xi}$ and $\mathsf{insertEff}_{\mathsf{p},\xi}$, become unnecessary and are removed from A_Λ, making the classical planning task P_Λ easier to be solved. For example, suppose that the preconditions of the *blocksworld* action schema `stack` are known, then the initial state is extended with literals, `(pre_holding_v1_stack)` and `(pre_clear_v2_stack)` and the associated actions $\mathsf{insertPre}_{\mathsf{holding_v1,stack}}$ and $\mathsf{insertPre}_{\mathsf{clear_v2,stack}}$ can be safely removed from the A_Λ action set without altering the *soundness* and *completeness* of the P_Λ compilation.

References

[Aineto *et al.*, 2018] Diego Aineto, Sergio Jiménez, and Eva Onaindia. Learning STRIPS action models with classical planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, pages 399–407. AAAI Press, 2018.

[Alcázar and Torralba, 2015] Vidal Alcázar and Alvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *ICAPS*, pages 2–6. AAAI Press, 2015.

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

[Fox and Long, 1998] Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.

[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.

[Helmert, 2009] Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.

[Kautz and Selman, 1999] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *IJCAI*, volume 99, pages 318–325, 1999.

[Kucera and Barták, 2018] Jirí Kucera and Roman Barták. LOUGA: learning planning operators using genetic algorithms. In *Pacific Rim Knowledge Acquisition Workshop, PKAW-18*, pages 124–138, 2018.

[Lotinac and Jonsson, 2016] Damir Lotinac and Anders Jonsson. Constructing hierarchical task models using invariance analysis. In *ECAI*, pages 1274–1282, 2016.

[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.

[Mourão *et al.*, 2012] Kira Mourão, Luke S. Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *Conference on Uncertainty in Artificial Intelligence, UAI-12*, pages 614–623, 2012.

[Muggleton and De Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

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

[Pereira and Meneguzzi, 2018] Ramon Fraga Pereira and Felipe Meneguzzi. Heuristic approaches for goal recognition in incomplete domain models. *arXiv preprint arXiv:1804.05917*, 2018.

[Rintanen, 2014] Jussi Rintanen. Madagascar: Scalable planning with SAT. In *International Planning Competition, (IPC-2014)*, 2014.

[Rintanen, 2017] Jussi Rintanen. Schematic invariants by reduction to ground invariants. In *AAAI*, pages 3644–3650, 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.

[Sreedharan *et al.*, 2018] Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, pages 518–526, 2018.

[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.

[Zhuo *et al.*, 2013] Hankz Hankui Zhuo, Tuan Anh Nguyen, and Subbarao Kambhampati. Refining incomplete planning domain models through plan traces. In *International Joint Conference on Artificial Intelligence, IJCAI-13*, pages 2451–2458, 2013.

[Zhuo, 2015] Hankz Hankui Zhuo. Crowdsourced action-model acquisition for planning. In *National Conference on Artificial Intelligence, AAAI-15*, pages 3439–3446, 2015.