Learning STRIPS action models from *state-invaraints*

Diego Aineto[1] , **Sergio Jiménez**[1] , **Eva Onaindia**[1]

[1]Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València. Valencia, Spain

{dieaigar,serjice,onaindia}@dsic.upv.es, miquel.ramirez@unimelb.edu.au

Abstract

1 Introduction

Classical planing is an interesting approach for learning STRIPS action models since it is flexible to different kinds of input knowledge (e.g., partially/fully observations of actions of plan executions as well as partially/fully observed intermediate states) [Aineto *et al.*, 2018]. This paper shows that this flexibility of the compilation approach goes one step further since it also allows learning from *state-invariants*, logic formulae that constrain the set of possible states of a given domain. The experimental results show that *state-invariants* boost the performance of the *classical planing* compilation for learning STRIPS action models.

2 Background

This section formalizes the *classical planning model* we follow in this work and the kind of *knowledge* that can be given as input to the task of learning STRIPS action models.

2.1 Classical planning with conditional effects

Let F be the set of propositional state variables (*fluents*) 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 contain conflicting values). Given L, let $\neg L = \{\neg l : l \in L\}$ be its complement. 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; $|s| = |F|$.

A *classical planning action* $a \in A$ has: a precondition $\mathsf{pre}(a) \in \mathcal{L}(F)$, a set of effects $\mathsf{eff}(a) \in \mathcal{L}(F)$, and a positive action cost $cost(a)$. The semantics of actions $a \in A$ is specified with two functions: $\rho(s, a)$ denotes whether action a is *applicable* in a state s and $\theta(s, a)$ denotes the *successor state* that results of applying action a in a state s. Then, $\rho(s, a)$ holds iff $\mathsf{pre}(a) \subseteq s$, i.e. if its precondition holds in s. The result of executing an applicable action $a \in A$ in a state s is a new state $\theta(s, a) = (s \setminus \neg\mathsf{eff}(a)) \cup \mathsf{eff}(a)$. Subtracting the complement of $\mathsf{eff}(a)$ from s ensures that $\theta(s, a)$ remains a well-defined state. The subset of action effects that assign a positive value to a state 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* of an action $a \in A$.

A *classical 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* and $cost(\pi) = \sum_{a \in \pi} cost(a)$ its *plan cost*. The execution of π on the initial state of P induces a *trajectory* $\tau(\pi, P) = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$ 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* $\tau(\pi, P)$ reaches a final state $G \subseteq s_n$, where all goal conditions are met. A solution plan is *optimal* iff its cost is minimal.

We also define *actions with conditional effects* because they are useful to compactly formulate our approach for *goal recognition with unknown domain models*. An action $a_c \in A$ with conditional effects is a set of preconditions $\mathsf{pre}(a_c) \in \mathcal{L}(F)$ and a set of *conditional effects* $\mathsf{cond}(a_c)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a_c)$ 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_c is applicable in a state s if $\rho(s, a_c)$ is true, and the result of applying action a_c in state s is $\theta(s, a_c) = \{s \setminus \neg\mathsf{eff}_c(s, a) \cup \mathsf{eff}_c(s, a)\}$ where $\mathsf{eff}_c(s, a)$ are the *triggered effects* resulting from the action application (conditional effects whose conditions hold in s):

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

2.2 The observation model

Given a planning problem $P = \langle F, A, I, G \rangle$, a plan π and a trajectory $\tau(\pi, P)$, we define the *observation of the trajectory* as an interleaved combination of actions and states that represents the observation from the execution of π in P. Formally, $\mathcal{O}(\tau) = \langle s_0^o, a_1^o, s_1^o \ldots, a_l^o, s_m^o \rangle$, $s_0^o = I$, and:

- The **observed actions** are consistent with π, which means that $\langle a_1^o, \ldots, a_l^o \rangle$ is a sub-sequence of π. The number of observed actions, l, ranges from 0 (fully unobserved action sequence) to $|\pi|$ (fully observed action sequence).
- The **observed states** $\langle s_0^o, s_1^o, \ldots, s_m^o \rangle$ is a sequence of possibly *partially observable states*, except for the initial state s_0^o, which is fully observed. A partially observable

state s_i^o is one in which $|s_i^o| < |F|$; i.e., a state in which at least a fluent of F is not observable. Note that this definition also comprises the case $|s_i^o| = 0$, when the state is fully unobservable. Whatever the sequence of observed states of $\mathcal{O}(\tau)$ is, it must be consistent with the sequence of states of $\tau(\pi, P)$, meaning that $\forall i, s_i^o \subseteq s_i$. The number of observed states, m, range from 1 (the initial state, at least), to $|\pi|+1$, and each *observed* states comprises $[1, |F|]$ fluents (the observation can still miss intermediate states that are *unobserved*).

We assume a bijective monotone mapping between actions/states of trajectories and observations [Ramírez and Geffner, 2009], thus also granting the inverse consistency relationship (the trajectory is a superset of the observation). Therefore, transiting between two consecutive observed states in $\mathcal{O}(\tau)$ 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. In other words, having an input observation $\mathcal{O}(\tau)$ does not imply knowing the actual length of π.

2.3 State-invaraints

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 compactly specifying sets of states. For instance, *state-constraints* in planning allow to specify the set of states where a given action is applicable, the set of states where a given *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 [Helmert, 2009] or making *satisfiability planning* and *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.

The formula $\phi_{I,A}^*$ represents the *strongest invariant* and exactly characterizes the set of all states reachable from I with the actions in A. For instance Figure 1 shows five clauses that define the *strongest invariant* for *blocksworld*. There are infinitely many strongest invariants, but they are all logically equivalent, and computing the strongest invariant is PSPACE-hard as hard as testing plan existence.

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

Figure 1: Example of the *strongest invariant* 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*, $\phi_1 = \neg on(block_A, block_B) \vee \neg on(block_A, block_C)$ is a mutex because $block_A$ can only be on top of a single block.

A *domain invariant* is an instance-independent invariant, i.e. holds for any possible initial state and 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 and others, 2017]. For instance, $\phi_2 = \forall x : (\neg handempty \vee \neg holding(x))$, is a *domain mutex* for the *blocksworld* because the robot hand is never empty and holding a block at the same time.

3 Deductive learning of planning action models

The task of learning action models is defined by $\Lambda = \langle \mathcal{M}_\prime, \Phi, \mathcal{O}(\tau) \rangle$:

- $\mathcal{M}_\prime$ is the **initial domain model** (set of action models). This set is *empty*, when learning from scratch, or *partially specified*, when some fragments of the action models are known a priori.
- Φ, a set of *state-invariants* that constraint the set of possible states in the given domain.
- $\mathcal{O}(\tau)$ is an observation of a trajectory $\tau(\pi, P)$ produced by the execution of an unknown plan π that reaches the goals $G \in G[\cdot]$ starting from the initial state I in P.

A *solution* to a learning task $\Lambda = \langle \mathcal{M}, \mathcal{O}(\tau) \rangle$ is a domain model $\mathcal{M}'$ that is consistent with the information of $\mathcal{M}$ and with the observed plan trace τ. This means that the action sequence (plan) that solves the planning problem $\langle s_0, s_n \rangle$ with $\mathcal{M}'$ along with the state trajectory induced by this plan is consistent with the input observatio $\mathcal{O}(\tau)$ and only travers states that satisfy the states invariants.

3.1 Learning of planning action models with classical planning

Because of the combinatorial nature of the search for a solution plan, the sooner unpromising nodes are pruned from the search the more efficient the computation of a solution plan. Constraints can be used to confine earlier the set of possible STRIPS action models and reduce then the learning hypothesis space. With regard to our compilation, *domain mutex* are useful to reduce the amount of applicable actions for programming a precondition or an effect for a given action schema. For example given the *domain mutex* $\phi = (\neg f_1 \vee \neg f_2)$ such that $f_1 \in F_v(\xi)$ and $f_2 \in F_v(\xi)$, we can redefine the corresponding programming actions for **removing** the *precondition* $f_1 \in F_v(\xi)$ from the action schema $\xi \in \mathcal{M}$ as:

$$\begin{aligned}
\mathsf{pre}(\mathsf{programPre}_{\mathsf{f_1},\xi}) =& \{\neg del_{f_1}(\xi), \neg add_{f_1}(\xi), mode_{prog}, pre_{f_1}(\xi), pre_{f_2}(\xi)\}\\
\mathsf{cond}(\mathsf{programPre}_{\mathsf{f_1},\xi}) =& \{\emptyset\} \rhd \{\neg pre_{f_1}(\xi)\}.
\end{aligned}$$

4 Evaluation

5 Conclusions

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.

[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.

[Ramírez and Geffner, 2009] Miquel Ramírez and Hector Geffner. Plan recognition as planning. In *International Joint conference on Artifical Intelligence, (IJCAI-09)*, pages 1778–1783. AAAI Press, 2009.

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

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