Learning STRIPS action models with classical planning

Diego García and **Sergio Jiménez** and **Eva Onaindia**
Departamento de Sistemas Informáticos y Computación
Universitat Politécnica de Valéncia.
Camino de Vera s/n. 46022 Valencia, Spain
{,serjice,onaindia}@dsic.upv.es

Abstract

This paper presents a novel approach for learning STRIPS action models from examples that compiles the inductive learning task into a classical planning task. Our compilation for this learning task is flexible to different amounts of available input knowledge; it accepts partially specified action models and what is more, the input learning examples can range from a set of plans (with their corresponding initial and final states) to only a set of initial and final states where no action is observed.

Introduction

Besides plan synthesis, planning action models are also useful for plan/goal recognition (Ramırez and Geffner 2010). In both tasks, off-the-shelf planners require reasoning about action models that correctly and completely capture the possible world transitions (Ghallab, Nau, and Traverso 2004; Geffner and Bonet 2013). Unfortunately, building such planning action models is complex, even for planning experts, so this knowledge acquisition bottleneck limits the potential of automated planning (Kambhampati 2007).

On the other hand, Machine Learning (ML) techniques are able to compute a wide range of different kinds of models from examples (Michalski, Carbonell, and Mitchell 2013). However, the application of inductive ML techniques for learning planning action models is not straightforward. First, the inputs to ML algorithms usually are sets of finite numeric vectors encoding the value of sets of objects features. The input for learning planning action models traditionally are sets of observations of plan executions (each with possibly different length). Second, the traditional output of off-the-shelf ML techniques is a scalar value (an integer, in the case of classification tasks, or a real value, in the case of regression tasks). In the case of learning STRIPS action models, the output is not a scalar but a model of the preconditions and the effects of each action that defines the possible state transitions in the given planning domain.

Learning STRIPS action models is a well-studied problem with sophisticated algorithms, like ARMS (Yang, Wu, and Jiang 2007), SLAF (Amir and Chang 2008) or LOCM (Cresswell, McCluskey, and West 2013) that do not require full knowledge of all the states traversed by the example plans. Motivated by recent advances on learning generative models with classical planning (Bonet, Palacios, and Geffner 2009; Segovia-Aguas, Jiménez, and Jonsson 2016; 2017) this paper introduces an innovative approach for learning classical planning action models that (1) it can be defined as a classical planning compilation and (2), it is flexible to different amounts of available input knowledge.

Background

This section defines the planning models used on this work.

Classical planning

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. $l = f$ or $l = \neg f$. A set of literals L represents a partial assignment of values to fluents (WLOG we 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 then a total assignment of values to fluents, i.e. $|s| = |F|$, so the size of the state space $2^{|F|}$. Explicitly including negative literals $\neg f$ in states simplifies subsequent definitions, but we often 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. Each action $a \in A$ has a set of literals $\mathsf{pre}(a) \in \mathcal{L}(F)$, called *preconditions*, a set of positive effects $\mathsf{eff}^+(a) \in \mathcal{L}(F)$, and a set of negative effects $\mathsf{eff}^-(a) \in \mathcal{L}(F)$. An action $a \in A$ is applicable in state s iff $\mathsf{pre}(a) \subseteq s$, and the result of applying a in s is a new state $\theta(s, a) = (s \setminus \mathsf{eff}^-(a)) \cup \mathsf{eff}^+(a)$.

A *classical planning problem* is a tuple $P = \langle F, A, I, G \rangle$, where I is an initial state and $G \in \mathcal{L}(F)$ is a goal condition. A *plan* for P is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that induces a state sequence $\langle s_0, s_1, \ldots, s_n \rangle$ 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 π *solves* P if and only if $G \subseteq s_n$, i.e. if the goal condition is satisfied following the application of π in I.

In this work we assume that the fluents in F are instantiated from predicates, as in PDDL (McDermott et al. 1998; Fox and Long 2003). There exists a set of predicates Ψ, each $p \in \Psi$ with an argument list of arity $ar(p)$. Given a set of objects Ω, the set of fluents F is then induced by assigning objects in Ω to the arguments of predicates in Ψ, 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.

Likewise, we assume that each action in A is instantiated from an STRIPS operator schema. Figure 1 shows the *stack* operator schema for a STRIPS blocksworld represented in PDDL.

```
(:action stack
  :parameters (?x1 ?x2)
  :precondition (and (holding ?x1) (clear ?x2))
  :effect (and (not (holding ?x1)) (not (clear ?x2))
               (clear ?x1) (handempty) (on ?x1 ?x2)))
```

Figure 1: Example of a *stack* operator schema for a STRIPS blocksworld represented in PDDL.

Let Ω_v be a new set of objects, $\Omega \cap \Omega_v = \emptyset$, that represent *variable names*. $|\Omega_v|$ is given by the action with the maximum arity in a planning frame. For instance, in a three-block blocksworld $\Omega = \{block_1, block_2, block_3\}$ and $\Omega_v = \{v_1, v_2\}$ because the operators `stack` and `unstack` are the ones with the maximum arity (two parameters each).

Let us define a new set of fluents F_v that results instantiating Ψ but using only the *variable objects* Ω_v. In 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 are now ready to define a STRIPS operator schema as a tuple $\xi = \langle head(\xi), pre(\xi), add(\xi), del(\xi)\rangle$:

- $head(\xi) = \langle name(\xi), pars(\xi)\rangle$, represents an operator *header* defined by its corresponding action name and a list of variables, $pars(\xi) \in \Omega_v^{ar(\xi)}$. The headers for the blocksworld operators 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 positive effects $add(\xi) \subseteq F_v$, and the negative effects $del(\xi) \subseteq F_v$ such that, $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$.

Classical planning with conditional effects

Our approach for leaning STRIPS action models is compiling this leaning task into a classical planning task with conditional effects. We use conditional effects because they allow us to compactly define actions whose effects depend on the current state. Many classical planners cope with conditional effects without compiling them away. In fact, the support of PDDL conditional effects was a requirement for participating at IPC-2014 (Vallati et al. 2015).

Now an action $a \in A$ has a set of literals $\mathsf{pre}(a) \in \mathcal{L}(F)$ called the *precondition* 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 state s if and only if $\mathsf{pre}(a) \subseteq s$, and the resulting set of *triggered effects* is

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

i.e. effects whose conditions hold in s. The result of applying a in s is a new state $\theta(s,a) = (s \setminus \mathsf{eff}^-(s,a)) \cup \mathsf{eff}^+(s,a)$, where $\mathsf{eff}^-(s,a)$ and $\mathsf{eff}^+(s,a)$ are the negative and positive effects in $\mathsf{eff}(s,a)$.

Learning STRIPS action models

Learning STRIPS action models from fully available input knowledge, i.e. a set of plans where the *pre-* and *post-states* of every action in a plan are available, is straightforward. In this case, the operators schema are derived lifting the literals that change between the pre and post-state of the corresponding action executions. Likewise, preconditions are derived lifting the minimal set of literals appearing in all the pre-states that correspond to the same operator.

This section formalizes more challenging tasks, for learning STRIPS action models, where less input knowledge is available. Next we formalize these learning tasks according to the available input knowledge.

Learning from labeled plans

This learning task is formalized as $\Lambda = \langle \Psi, \Pi, \Sigma\rangle$:

- Ψ, the set of predicates that define the abstract state space of a given planning domain. This set includes the predicates for defining the headers of the operators schema.
- $\Pi = \{\pi_1, \ldots, \pi_\tau\}$, the given set of example plans.
- $\Sigma = \{\sigma_1, \ldots, \sigma_\tau\}$, a set of labels s.t. each plan π_t, $1 \leq t \leq \tau$, has a label $\sigma_t = (s_0^t, s_n^t)$ where s_n^t is the state resulting from executing π_t starting from the state s_0^t.

A solution to the learning task Λ is a set of operator schema Ξ (one schema for each operator header) compliant with the predicates in Ψ, the example plans Π, and their labels Σ.

Learning from initial/final states

Here we reduce the amount of input knowledge provided to the learning task. Now $\Pi = \{\pi_1, \ldots, \pi_\tau\}$ is replaced by $\Pi' = \{|\pi_1|, \ldots, |\pi_\tau|\}$ i.e. Π' that does not contain a set of plans but the number of actions of each plan, so the learning task is redefined as $\Lambda' = \langle \Psi, \Pi', \Sigma\rangle$. While the previous learning task, Λ, corresponds to watching an agent acting in the world, this new learning task Λ' can be understood as watching only the results of its actions executions knowing the number of diferent actions performed by the agent.

Finally, we can go a step further and redefine a third learning task $\Lambda'' = \langle \Psi, \Sigma\rangle$ that corresponds to watching only the results of the plan executions. In this case a solution to the Λ'' learning task is a set of operator schema Ξ that is compliant only with the predicates in Ψ, and the given set of initial and final states Σ.

In these two cases, a solution must not only synthesize the action models but also the actions that produced the given labels (this information is no longer given in the learning examples).

Learning STRIPS action models with classical planning

Our approach for addressing a learning task Λ, Λ' or Λ'', is to compile it into a classical planning task P_Λ, $P_{\Lambda'}$ or $P_{\Lambda''}$. The intuition behind these compilations is that a solution to the resulting classical planning task is a sequence of actions that:

1. Programs the action model i.e. determines the literals in the sets $pre(\xi)$, $del(\xi)$ and $add(\xi)$ for each $\xi \in \Xi$.
2. Validates the programmed action model in the given set of labels $\Sigma = \{\sigma_1, \ldots, \sigma_\tau\}$ i.e., for every $1 \leq t \leq \tau$, uses the programmed action model Ξ to produce a final state s_n^t starting from its corresponding initial state s_0^t.

To formalize these compilations we define $1 \leq t \leq \tau$ classical planning instances $P_t = \langle F, A, I_t, G_t \rangle$, that belong to the same planning frame (share the same fluents and actions and differ only in the initial state and goals). The set of fluents F is built instantiating the predicates in Ψ with the objects appearing in the labels. Formally $\Omega = \{o | o \in s_0^t \cup s_n^t \cup \pi_t, 1 \leq t \leq \tau\}$. The set of actions is empty $A = \emptyset$, is the aim of the learning tasks adressed in the paper. Finally the initial state I_t is given by the state $s_0^t \in \sigma_t$ while goals G_t are defined by the state $s_n^t \in \sigma_t$.

Now we are ready to define the compilation for learning STRIPS action models using classical planning with conditional effects. Given a learning task $\Lambda'' = \langle \Psi, \Sigma \rangle$ the compilation outputs a classical planning task $P_{\Lambda''} = \langle F_\Lambda, A_\Lambda, I_\Lambda, G_\Lambda \rangle$ where:

- F_Λ extends F with:
  - Fluents representing the programmed action model: $pre_f(\xi)$, $del_f(\xi)$ and $add_f(\xi)$ for every $f \in F_v$ and $\xi \in \Xi$.
  - Fluents $\{test_t\}_{1 \leq t \leq \tau}$, indicating the example where the programmed model is currently being validated.
  - Fluents $mode_{pre}$, $mode_{eff}$ and $mode_{val}$ indicating whether currently the preconditions of the action schema are being programmed, the effects of the action schema are being programmed or the programmed action models are being validated.
- I_Λ, contains the fluents from F that encode s_0^1, every fluent $pre_f(\xi) \in F_\Lambda$ and $mode_{pre}$.
- $G_\Lambda = \{test_t\}$,$1 \leq t \leq \tau$, indicates that the programmed action model is validated in all the learning examples.
- A_Λ contains actions of three types:
  1. Actions for programming:
     - *Precondition* $f \in F_v$ *in the action schema* $\xi \in \Xi$:

$$\begin{aligned}\mathsf{pre}(\mathsf{programPre}_{\mathsf{f},\xi}) =& \{pre_f(\xi), \neg prog2, \neg exec\},\\ \mathsf{cond}(\mathsf{programPre}_{\mathsf{f},\xi}) =& \{\emptyset\} \rhd \{\neg pre_f(\xi)\}.\end{aligned}$$

     - *Negative effect* $f \in F_v$ *in the action schema* $\xi \in \Xi$:

$$\begin{aligned}\mathsf{pre}(\mathsf{programDel}_{\mathsf{f},\xi}) =& \{pre_f(\xi), \neg del_f(\xi), \neg add_f(\xi), \neg exec\},\\ \mathsf{cond}(\mathsf{programDel}_{\mathsf{f},\xi}) =& \{\emptyset\} \rhd \{del_f(\xi),\\ & \{\emptyset\} \rhd \{prog2\}.\end{aligned}$$

     - *Positive effect* $f \in F_v$ *in the action schema* $\xi \in \Xi$:

$$\begin{aligned}\mathsf{pre}(\mathsf{programAdd}_{\mathsf{f},\xi}) =& \{\neg pre_f(\xi)), \neg del_f(\xi)), \neg add_f(\xi), \neg exec\},\\ \mathsf{cond}(\mathsf{programAdd}_{\mathsf{f},\xi}) =& \{\emptyset\} \rhd \{add_f(\xi),\\ & \{\emptyset\} \rhd \{prog2\}.\end{aligned}$$

  2. Actions for applying an already programmed operator schema $\xi \in \Xi$ bounding it with objects $\omega \subseteq \Omega^{ar(\xi)}$

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

     For instance, these actions define that if an operator is programmed with precondition $holding(v_1) \in F_v$ it implies that $holding(block_1) \in F$ has to be true in the current state if the operator binds variable object $v_1 \in \Omega_v$ with object $block_1 \in \Omega_v$. The operator binding is done implicitly, i.e. variables in $pars(\xi)$ are bound to the objects in ω appearing in the same position.
  3. Actions for changing the active example where the action model is currently being validated.

$$\begin{aligned}\mathsf{pre}(\mathsf{validate_i}) =& G_i \cup \{test_j\}_{j \in 1 \leq j < i} \cup \{exec\},\\ \mathsf{cond}(\mathsf{validate_i}) =& \{\emptyset\} \rhd \{test_i\}.\end{aligned}$$

Lemma 1. *Any classical plan π that solves P_Λ induces a valid action model that solves the learning task Λ.*

Proof sketch. Once an operator schema is programmed it cannot be modified and can only be applied. The only way of achieving a *test* fluent is by applying a sequence of programmed operator schema until achieving the goal state defined by its associated label starting from the initial state of the corresponding label. If this is done for all the labels (all the input examples) it means that the programmed action model is compliant with the learning input knowledge and hence, it is a solution to the action model learning task. □

The compilation is valid for partially specified action models since known preconditions and effects (fluents $pre_f(\xi)$, $del_f(\xi)$ and $add_f(\xi)$) can be part of the initial state I_Λ. The approach allows also transfer learning where the action model generated for a given sub-task is encoded as *already programmed* for learning new action models in more challenging tasks.

Learning action models from example plans using a classical planner

The compilation can be extended to the learning scenario defined by Λ and Λ' in which a set of plans Π (or only its lengths in the case of Λ') is available. Each plan $\pi_i \in \Pi, 1 \leq i \leq t$, is a solution to the corresponding classical planning instance $P_i = \langle F, A, I_i, G_i \rangle$ defined above. The compilation extensions are:

- F_Λ includes the new set of fluents $F_\Pi = \{plan(name(\xi), j, \Omega^{ar(\xi)})\}$ for encoding the j steps of the $1 \leq i \leq t$ plans in Π with $F_{\Pi_i} \subseteq F_\Pi$ the fluents encoding the plan corresponding to the i^{th} example (only for the Λ case). In addition fluents at_j and $next_{j,j_2}$, $1 \leq j < j2 \leq n$, represent the plan step where the programmed action model is validated (n is the max length of a plan in Π).

- I_Λ is extended with the fluents from F_{Π_1} that encode the plan $\pi_1 \in \Pi$ for solving P_1, and the fluents at_1 and $\{next_{j,j_2}\}$, $1 \leq j < j2 \leq n$, for indicating the plan step where to start validating the programmed action model. Goals G_Λ are like in the original compilation.
- With respect to the actions in A_Λ,

1. The actions for programming the preconditions/effects of a given operator schema are the same.
2. The actions for applying an operator schema have an extra precondition $f \in F_{\Pi_i}$ that encodes the current plan step and extra conditional effect $\{at_j\} \rhd \{\neg at_j, at_{j+1}\}_{\forall j \in [1,n]}$ for advancing the plan step.
3. The actions for changing the active test have an extra precondition, $at_{|\Pi_i|}$, to indicate that we simulated the full current plan Π_i and extra conditional effects to load the next plan Π_{i+1} where to validate the programmed action model:

$$\{f\} \rhd \{\neg f\}_{f \in F_{\Pi_i}},$$
$$\{\emptyset\} \rhd \{f\}_{f \in F_{\Pi_i+1}},$$
$$\{\emptyset\} \rhd \{\neg at_{|\pi_i|}, at_1\}.$$

Evaluation

Learning action models from example plans

The performance of our learning approach is evaluated for different degrees of available input knowledge and using different sources for collecting this input knowledge. In all the cases we assess the performance of our learning approach using the cardinality of the *symmetric difference* sets that are computed between the set of preconditions, del and add effects (1), in the learned model and (2), in the actual models. In all the experiments the compilation is solved using the SAT-based classical planner MADAGASCAR (Rintanen 2014).

Table 1 shows the mean error and standard deviation of the learned models with respect to the actual action models when (1) using *hand-picked* examples, (2) examples collected using the classical *planner* FAST-DOWNWARD (Helmert 2006) and (3) examples collected *randomly*. The standard deviation provides a measure of how this error is distributed among the different operators in the domain. If this deviation is 0 it means that is equally distributed in all the domain operators.

Learning action models from example states

TBD.

Related work

Conclusions

This paper presents a novel approach for learning classical planning action models from minimal input knowledge and using exclusively existing classical planners. Learning action models from examples allows the reformulation of a domain theory. An interesting research direction is the study of domain reformulation using features that allow more compact solutions like the *reachable* or *movable* features in the Sokoban domain.

Last but not least, collecting *informative* examples for learning planning action models is challenging. Planning actions include preconditions that are only satisfied by specific sequences of actions, and often, with a low probability of being chosen by chance (Fern, Yoon, and Givan 2004). In addition, motivated by the success of recent algorithms for exploring planning tasks (**?**), we do not assume that a learning set of plans is given apriori but instead, we autonomously collect the learning examples.

References

Amir, E., and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research* 33:349–402.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *ICAPS*.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using locm. *The Knowledge Engineering Review* 28(02):195–213.

Fern, A.; Yoon, S. W.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *ICAPS*, 191–199.

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Geffner, H., and Bonet, B. 2013. A concise introduction to models and methods for automated planning.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)* 26:191–246.

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

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl-the planning domain definition language.

Michalski, R. S.; Carbonell, J. G.; and Mitchell, T. M. 2013. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*, 1121–1126.

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

Table 1: Mean error and standard deviation of the learned models when using hand-picked examples and examples collected using the classical planner Fast-Downward.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2016. Hierarchical finite state controllers for generalized planning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 3235–3241. AAAI Press.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2017. Generating context-free grammars using classical planning. In *International Joint Conference on Artificial Intelligence*.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3):90–98.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2-3):107–143.