Bootstrapping classical planning action models

Guillem Francés
Information and Communication Technologies
Universitat Pompeu Fabra
Roc Boronat 138, 08018 Barcelona, Spain
@upf.edu

Sergio Jiménez
Computing and Information Systems
University of Melbourne
Parkville, Victoria 3010, Australia
sjimenez@unimelb.edu.au

Nir Lipovetzky
Computing and Information Systems
University of Melbourne
Parkville, Victoria 3010, Australia
@unimelb.edu.au

Miguel Ramírez
Computing and Information Systems
University of Melbourne
Parkville, Victoria 3010, Australia
@unimelb.edu.au

Abstract

This paper presents a novel approach for learning classical planning action models from minimal input knowledge and using exclusively existing classical planners. First, the paper defines a classical planning compilation to learn action models from examples. Second, the paper explains how to collect informative examples using a classical planner based on pure exploratory search.

Introduction

Off-the-shelf planners reason about action models that correctly and completely capture the possible world transitions (**?**). Unfortunately building such models is complex even for planning experts limiting the potential of automated planning (**?**).

In Machine Learning (ML) models are not hand-coded but computed from examples (**?**). Applying off-the-shelf ML to learning planning action models is not straightforward though. On the one hand the inputs of ML algorithms are usually objects features encoded as a finite set of values. Planning tasks include actions and are closer to the representation of procedures and behaviours than to object representation. On the other hand, 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). For the task of learning classical planning actions the output is not a scalar but a declarative generative model of the state transitions. Finally the collection of *informative* examples for learning planning action models is complex. Planning actions include preconditions that are only satisfied by specific sequences of actions and often with a low probability of being chosen by chance (**?**).

The learning of action models for classical planners is a well-studied problem and there are sophisticated algorithms for this learning task, like ARMS (**?**) or LOCM (**?**) that do not require the full knowledge of the states traversed by example plans. This work aims going one step beyond these approaches and address learning classical planning action models when the only available knowledge is pairs of initial and final states.

Motivated by recent advances on learning generative models using classical planners (**?**) and on the effective exploration of planning state spaces (**?**), this paper introduces an innovative approach for learning classical planning action models. The contribution of the work is two-fold:

1. An inductive learning algorithm that minimizes the required input knowledge, that is flexible to different levels of available input knowledge, and that can be defined as a classical planning compilation.
2. A method for autonomously collect *informative* examples for action model learning using an exploration-based classical planner.

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 a total assignment of values to fluents, i.e. $|s| = |F|$, so the number of states is $2^{|F|}$.

Under this formalism, 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 the *preconditions*, a set of positive effects $\mathsf{add}(a) \in \mathcal{L}(F)$ and a set of negative effects $\mathsf{del}(a) \in \mathcal{L}(F)$. 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 $\theta(s, a) = (s \setminus \neg\mathsf{del}(a)) \cup \mathsf{add}(a)$.

Given a planning frame $\Phi = \langle F, A\rangle$, 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 i such that $1 \le i \le n$, 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.

We assume that fluents are instantiated from predicates, as in PDDL (**?**). Specifically, 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. $F = \{p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}\}$ where, given a set X, X^n is the n-th Cartesian power of X.

Likewise we assume actions in A are instantiated from operator schema as follows. Let us define $\Omega_v = \{var_1, \dots, var_v\}$ as a new set of objects, $\Omega_v \cap \Omega = \emptyset$, representing variable names. The number of *variable* objects, $|\Omega_v|$, is given by the action with the maximum arity. For instance, in the blocksworld $\Omega_v = \{var_1, var_2\}$ since action $stack$, Figure 1, (and $unstack$) has two parameters. Let us define a new set of fluents, $F_v \cap F = \emptyset$, that result instantiating Ψ but using only the variable objects in Ω_v.

```
(: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 the *stack* planning action schema from the blocksworld as represented in PDDL.

Now we are ready to define an operator schema $\xi \in \Xi$ as:

- The operator *header*: a fluent $header(name(\xi), pars(\xi))$ built using the action name, e.g. $name(\xi) \in \{stack, unstack, pickup, putdown\}$ for the blocksworld, and a list of variables, $pars(\xi) \in \Omega_v^{ar(\xi)}$.
- The operator *body* comprises: 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

Conditional effects make it possible to repeatedly refer to the same action even though their precise effects depend on the current state. In this case each 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 $\theta(s, a) = (s \setminus \neg\mathsf{eff}(s, a)) \cup \mathsf{eff}(s, a)$.

Learning classical planning action models

This section formalizes the different learning tasks addressed in the paper according to the available amount of input knowledge.

The largest possible amount of available input knowledge is learning from the *pre-* and *post-states* of every action execution. This learning task is straightforward since the dynamics of a given operator are derived lifting the literals that change between the pre and post-state of the corresponding action execution. The preconditions are the minimal set of literals appearing in all the pre-states corresponding to the same operator.

Learning classical planning action models from plans This 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 describing the actions header.
- $\Pi = \{\pi_1, \dots, \pi_t\}$ is the given set of example plans,
- $\Sigma = \{\sigma_1, , \dots, \sigma_t\}$ is a set of labels s.t. each plan π_i, $1 \leq i \leq t$, has an associated label $\sigma_i = (s_i, s_i')$ where s_i' is the state resulting from executing π_i starting from the state s_i.

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

Learning classical planning action models from states As explained in this paper we aim to reduce the amount of input knowledge provided to the learning task so we redefine it as $\Lambda' = \langle \Psi, \Sigma \rangle$.

A solution to the Λ' learning task is a set of operator schema Ξ that is compliant with the predicates in Ψ, and the given set of initial and final states.

Learning action models from states using a classical planner

Our approach for addressing $\Lambda' = \langle \Psi, \Sigma \rangle$ is compiling this learning task into a classical planning task P_Λ. The intuition behind the compilation is that a solution to P_Λ is a sequence of actions that first, programs the action action model (i.e. the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ for each $\xi \in \Xi$) and then, sequentially, validates the programmed action model in the given set of labels Σ, one after the other.

To formalize the compilation we first define t classical planning instances $P_1 = \langle F, A, I_1, G_1 \rangle, \dots, P_t = \langle F, A, I_t, G_t \rangle$, that belong to the same planning frame $\Phi = \langle F, A \rangle$ (i.e. share the same fluents and actions and differ only in the initial state and goals). The sets of actions is empty $A = \emptyset$ while the set of fluents F is built instantiating the predicates in Ψ with the objects in Ω, where Ω, is the set of objects that appear in the states in Σ, i.e., $\Omega = \{o | o \in s_i \cup s_i', 1 \leq i \leq t\}$. Finally the initial state I_i, $1 \leq i \leq t$, is given by the state $s_i \in \sigma_i$ and the goals are defined by the state $s_i' \in \sigma_i$.

Now we are ready to define the compilation for learning action models using an off-the-shelf classical planner. Given a learning task $\Lambda' = \langle \Psi, \Sigma \rangle$ the compilation outputs a classical planning task $P_\Lambda = \langle F_\Lambda, A_\Lambda, I_\Lambda, G_\Lambda \rangle$:

- F_Λ extends F with:

- Fluents representing the programmed action model: $header(\xi)$, $pre_f(\xi)$, $del_f(\xi)$ and $add_f(\xi)$ where $f \in F_v$ and $\xi \in \Xi$.
- Fluents $\{test_i\}_{1 \leq i \leq t}$, indicating the example where the programmed model is currently being validated.

- I_Λ, contains the fluents from F that encode the initial state $s_1 \in P_1$ and the header of the operators we want to learn.
- $G_\Lambda = \{test_i\}$,$1 \leq i \leq t$, indicates that the programmed action model is validated in all the examples.
- A_Λ replaces the actions in A with actions of three types:

1. Actions for programming:
   - A *precondition* $f \in F_v$ in the action schema $\xi \in \Xi$:

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

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

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

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

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

2. Actions for applying an already programmed operator schema $\xi \in \Xi$ and that is bound with objects $v' \in \Omega^{ar(\xi)})$

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

3. Actions for changing the active example where the action model is currently being validated.

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

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 executed. The only way of achieving a test fluent is by applying a sequence of programmed operator schema until achieving the goal state in its associated label. If this is done for all the input examples it means that the programmed action model is compliant with the learning input knowledge and hence, that solves the action model learning task. □

Interestingly this compilation is valid for partially specified action models since the known preconditions and effects (fluents $pre_f(\xi)$, $del_f(\xi)$ and $add_f(\xi)$) can be part of the initial state I_Λ. With this regard the approach allows also transfer learning in which we generate the action model for a given sub-task and then encode this model as already programmed actions for learning new or more challenging action models.

Learning action models from plans using a classical planner

The compilation can be extended to the learning scenario in which a set of action plans Π is available s.t., each plan $\pi_i \in \Pi, 1 \leq i \leq t$, is a solution to the corresponding classical planning instance $P_i = \langle F, A, I_i, G_i \rangle$:

- F_Λ includes the set of fluents $F_{plan} = \{plan(name(\xi), i, \Omega^{ar(a)})\}$ for encoding the plans in Π. Fluents at_i and $next_{i,i_2}$, $1 \leq i < i2 \leq n$, indicate the plan step where the programmed action model is being validated and n is the max length of a plan in Π.
- I_Λ is extended with the fluents from F_{plan} that encode the plan $\pi_1 \in \Pi$ for solving P_1, and the fluents at_1 and $\{next_{i,i_2}\}$, $1 \leq i < i2 \leq n$, for indicating the plan step where to start validating the programmed model. Goals are like in the original compilation.
- With respect to the actions,

1. The actions for programming the preconditions and effects of a given operator schema are the same.
2. The actions for applying an operator schema have an extra precondition $f \in F_{plan}$ s.t. f encodes the current plan step and extra conditional effect $\{at_i\} \rhd \{\neg at_i, at_{i+1}\}_{\forall i \in 1 \leq i < n}$ 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 and extra conditional effects to load the next plan where to validate the programmed action model:

$$\begin{aligned}&\{\emptyset\} \rhd \{\neg plan(\xi, k, v\}_{1 \leq k < |\Pi_i|},\\ &\{\emptyset\} \rhd \{plan(\xi, k, v\}_{1 \leq k < |\Pi_{i+1}|},\\ &\{\emptyset\} \rhd \{\neg at_{|\pi_i|}, at_1\}.\end{aligned}$$

Generating informative learning examples

Observation: If a predicate is not appearing either in the initial not in the final state it will not appear in the learned action model.

Evaluation

We evaluate 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, in the learned model and in the actual models. This performance metric is weighted by the size of the sets so a 100% value means that this set was learned perfectly and 0% means that no correct learning was accomplished.

The performance of our learning approach is evaluated for (1) different amounts of initial knowledge available and (2) using different sources for collecting this initial knowledge.

	Prec		Del		Add	
	Avg	Var	Avg	Var	Avg	Var
Blocks	5.0	75.0	0.0	0.0	0.0	0.0
Grippper	65.5	150.6	0.0	0.0	5.1	14.4
Miconic	41.3	482.8	0.0	0.0	6.3	41.5
Visitall	50.0	0.0	0.0	0.0	0.0	0.0

Related work

Conclusions

Learning action models from examples allows the reformulation of a domain theory. Using classical planning focus the learning task in the more succinct action models given that classical planners have a preference for short plans.

References

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.

Francs, G.; Ramrez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely declarative action representations are overrated: Classical planning with simulators. In *International Joint Conference on Artificial Intelligence*.

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

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*.

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

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

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.