STRIPS Action Discovery

Alejandro Suárez-Hernández[1]**, Javier Segovia-Aguas**[1]**, Carme Torras**[1]**, Guillem Alenyà**[1]

[1]Institut de Robòtica i Informàtica Industrial, CSIC-UPC.

Llorens i Artigas 4-6, 08028 Barcelona, Spain.

`{asuarez, jsegovia, torras, galenya}@iri.upc.edu`

Abstract

The problem of specifying high-level knowledge bases for planning becomes a hard task in realistic environments. This knowledge is usually handcrafted and is hard to keep updated, even for system experts. Recent approaches have shown the success of classical planning at synthesizing action models even when all intermediate states are missing. These approaches can synthesize action schemas in Planning Domain Definition Language (PDDL) from a set of execution traces each consisting, at least, of an initial and final state. In this paper, we propose a new algorithm to unsupervisedly synthesize STRIPS action models with a classical planner when action signatures are unknown. In addition, we contribute with a compilation to classical planning that mitigates the problem of learning static predicates in the action model preconditions, exploits the capabilities of SAT planners with parallel encodings to compute action schemas and validate all instances. Our system is flexible in that it supports the inclusion of partial input information that may speed up the search. We show through several experiments how learnt action models generalize over unseen planning instances.

Introduction

Most work in Artificial Intelligence planning assumes models are given as input for *plan synthesis* (Ghallab, Nau, and Traverso 2004). Nevertheless, the acquisition of such models in complex environments becomes a hard task even for system experts limiting the potential applications of planning (Kambhampati 2007). Research in web-service composition (Carman, Serafini, and Traverso 2003) and work-flow management (Blythe, Deelman, and Gil 2004) as planning is an example of bottleneck for generating the input specification.

Kambhampati (2007) (Kambhampati 2007) proposes in his work a shallow or approximate planning domain representation such that it can be criticized and/or refined with experiences. In that sense, the problem of learning action models consists on inferring, synthesizing or refining action schemas that can be used for planning on new tasks in the same environment. However, frameworks for learning action models varies from their inputs, action granularity, techniques, and output structures (Arora et al. 2018). On the one hand, most frameworks require input traces with the action signatures (i.e. action names and parameters), ob-
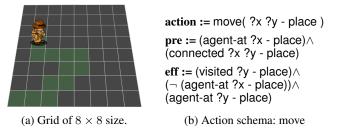

action := move(?x ?y - place)

pre := (agent-at ?x - place)∧
(connected ?x ?y - place)

eff := (visited ?y - place)∧
(¬ (agent-at ?x - place))∧
(agent-at ?y - place)

(a) Grid of 8×8 size.　　　　(b) Action schema: move

Figure 1: *visit-all* example with the action schema.

jects and predicates such as ARMS (Yang, Wu, and Jiang 2007), SLAF (Amir and Chang 2008), and STRIPS-based compilations (Aineto, Jiménez, and Onaindia 2018). On the other hand, the LOCM family of algorithms (Cresswell, McCluskey, and West 2009; Cresswell and Gregory 2011; Gregory and Cresswell 2015), generate object-centric planning models so they only need sequences of grounded actions. The action learner methods (Amado et al. 2018) can produce (without guarantees) a set of PDDL actions given a set of image transitions. The aim of these frameworks is to learn action models like the one in Figure 1b, that represents the action to move the robot between adjacent locations. Figure 1a shows an instance of *visitall*, where the agent starts at the bottom-left corner and must visit all cells in a grid. Learning action models consists in automatically generating the domain that rules any problem in a given environment, e.g. the move action with its preconditions and effects in the *visitall* domain.

Motivated by recent advances on unsupervised learning approaches to apply planning in the latent space (Asai and Fukunaga 2018; Amado et al. 2018), unsupervised generalized model synthesis with classical planners (Segovia-Aguas, Jiménez, and Jonsson 2017), and learning action models from minimal observability (Aineto, Celorrio, and Onaindia 2019), we present in this paper a novel compilation to classical planning for learning STRIPS action schemas like the one in Figure 1b with an unsupervised algorithm. The input to the compilation is a set of goal-oriented examples, i.e. a set of planning instances with their set of objects, predicates, initial states and goal conditions. Then, action schemas are induced from the plan that solves the compiled

planning problem. These are validated with the input examples in parallel, and evaluated on new instances to test their generalization. Moreover, these approaches can work with approximate models where a partial action schema is included as prior domain knowledge, but new actions can be learnt, validated on new evidences, and refine the domain model. The main contributions of this paper are:

- a STRIPS-based compilation to learn *action signatures* from minimal information with a SAT-based planning system,

- a method to learn static predicates by removing preconditions of action models instead of being programmed,

- new actions in the compilations adapted to validate input examples in parallel with the learnt action models,

- an algorithm to explore learning tasks in the bounded space of configurations, and

- an empirical evaluation of action schemas convergence in regard to the increasing number of input examples.

In the next section we introduce classical planning with conditional effects, action schemas in STRIPS, and formalize the problem of learning these actions with classical planning. Then, we formalize our task representation and its compilation to classical planning, followed by the configuration algorithm that generates the compilations. In the Experiments Section, we use the configuration algorithm to run experiments on learning and validating action models. Finally, we review related work and conclude the paper with some future work direction.

Preliminaries

In this section we introduce some preliminar notation of classical planning with conditional effects, and action schemas when defined in STRIPS language (Fikes and Nilsson 1971) and how the problem of learning action schemas have been usually addressed.

Classical planning with conditional effects

A *classical planning problem* is defined by a 4-tuple $P = \langle F, A, I, G \rangle$, where F is the set of fluents, A is the set of actions, $I \subseteq F$ is the initial state and $G \subseteq F$ a goal condition.

One extension of planning actions A, is to describe their effects conditionally affected by the current state s. Therefore, we formally define the state space $S = 2^{|L|}$ in terms of literals L, where a literal $l \in L$ is a valuation of a fluent $f \in F$, i.e. $l = f$ or $l = \neg f$. We assume $|s| = |L|$, L does not assign conflicting values and $\neg L = \{\neg l : l \in L\}$ to be the complement of L.

Then, an action $a \in A$ with conditional effects is defined as $a = \langle \mathsf{pre}(a), \mathsf{ce}(a) \rangle$, where $\mathsf{pre}(a)$ are the preconditions and $\mathsf{ce}(a)$ is the set of conditional effects. We say that an action $a \in A$ is *applicable* in state $s \in S$ iff the preconditions hold in that state, i.e. $\mathsf{pre}(a) \subseteq s$. Moreover, we denote $C \triangleright E \in \mathsf{ce}(a)$ to the sets of literals that correspond to the condition C and the effects E, and the whole set of *triggered effects* is defined as $\mathsf{eff}(s,a) = \cup_{C \triangleright E \in \mathsf{ce}(a), C \subseteq s} E$. Thus, the transition function is defined

as $\theta(s,a) = (s \setminus \mathsf{eff}^-(s,a)) \cup \mathsf{eff}^+(s,a)$, where $\mathsf{eff}^-(s,a)$ and $\mathsf{eff}^+(s,a)$ correspond to the negative and positive effects in $\mathsf{eff}(s,a)$ to apply in the current state s.

A solution to a planning problem is a *plan* defined as a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ that applied in the initial state $s_0 = I$ generates a sequence of $n + 1$ states $s_0, \ldots, s_n$ where the goal condition holds in the last one, i.e. $G \subseteq s_n$. For a plan to be valid, each action must be applicable in the corresponding state $\mathsf{pre}(a_i) \subseteq s_{i-1}$.

Action schemas and grounding in STRIPS

The aim of this work is to synthesize action schemas from demonstrations in STRIPS language (Fikes and Nilsson 1971), which is a subset of PDDL. Thus, for simplicity of the paper we represent the tasks in PDDL with negations, disjunctive preconditions and conditional effects, so that any off-the-shelf (expressive enough) planning system can be used, i.e. Madagascar SAT-based planner (Rintanen 2014).

We denote a *PDDL task* as $P_{\mathsf{PDDL}} = \langle \mathcal{P}, \mathcal{A}, \Sigma, I, G \rangle$ where $\mathcal{P}$ and $\mathcal{A}$ are the set of predicates and action schemas respectively, Σ is the set of objects, I is the initial state and G is the goal condition.

Let us define X as the possibly empty set of parameters that describe the parameterization of predicates and action schemas. Then, an *action schema* $\alpha[X]$ is defined as $\langle \mathsf{n}_\alpha, X, \mathsf{pre}_\alpha, \mathsf{add}_\alpha, \mathsf{del}_\alpha \rangle$ where n_α is the name and X is the set of parameters, both describe the header of the action, also known as action signature. Then, the body is described by preconditions, positive and negative effects.

Definition 1 (Well-defined Action Schema). *The action schema is well-defined if and only if the set of parameters used in the body is equal to the parameters of the action signature, i.e.* $X = \{Y : \forall p[Y] \in \{\mathsf{pre}_\alpha \cup \mathsf{add}_\alpha \cup \mathsf{del}_\alpha\}\}$.

Example 1 (Schema). *In Figure 1b we have an example where the header is described with a name* $\mathsf{n}_{\mathsf{move}} = \mathsf{move}$ *and a set of parameters* $X = \{x, y\}$ *both of type place. The body of the action schema consist of three sets of predicates, the set of preconditions* $\mathsf{pre}_{\mathsf{move}} = \{\mathsf{agent\text{-}at}[x], \mathsf{connected}[x,y]\}$, *the set of positive effects* $\mathsf{add}_{\mathsf{move}} = \{\mathsf{visited}[y], \mathsf{agent\text{-}at}[y]\}$ *and the negative effects* $\mathsf{del}_{\mathsf{move}} = \{\mathsf{agent\text{-}at}[x]\}$. *This action schema is well-defined, since the union of all predicate parameters in the body is* $\{x, y\}$ *which is equal to* X.

The grounding of predicates $p[X] \in \mathcal{P}$ and action schemas $\alpha[X] \in \mathcal{A}$ is the valid assignment of objects $\omega \in \Sigma^{|X|}$ to their parameters, where $\Sigma^{|X|}$ is the $|X|$-th Cartessian product of Σ objects. Therefore, the set of propositional variables or fluents is $F = \{p[\omega] : p[X] \in \mathcal{P}, \omega \in \Sigma^{|X|}\}$ which means, for every predicate there is a parameter assignment of $|X|$ objects. Thus, grounded actions are $A = \{\alpha[\omega] : \alpha[X] \in \mathcal{A}, \omega \in \Sigma^{|X|}\}$.

Example 2 (Grounding). *When grounding Example 1, if the set of objects is* $\Sigma = \{\mathsf{p1}, \mathsf{p2}\}$, *which are two different places in visitall domain, the predicates and actions are grounded as follows:* $F = \{\mathsf{at}[\mathsf{p1}], \mathsf{at}[\mathsf{p2}], \mathsf{v}[\mathsf{p1}], \mathsf{v}[\mathsf{p2}], \mathsf{conn}[\mathsf{p1}, \mathsf{p1}], \mathsf{conn}[\mathsf{p1}, \mathsf{p2}], \mathsf{conn}[\mathsf{p2}, \mathsf{p1}], \mathsf{conn}[\mathsf{p2}, \mathsf{p2}]\}$,

and the actions are $A = \{\mathsf{m[p1, p1]}, \mathsf{m[p1, p2]}, \mathsf{m[p2, p1]}, \mathsf{m[p2, p2]}\}$. *We have simplified notation, such that* agent-at *is* at, visited *is* v, connected *is* conn, *and* move *is* m. *Once predicates and actions are grounded, the set of reachable states will depend on the initial state of each instance, even though there are some other grounding strategies (Gnad et al. 2019).*

Learning action schemas from plan traces

The problem of learning action schemas is usually formalized as a function that minimizes a cost in the current model from observing an agent acting in the environment. In this section we describe these observations in plan traces, and formalize the inputs/outputs for the task of learning action models.

Plan traces   There are many representation mechanisms for learning action models (Arora et al. 2018), but all these systems include at least a set of plan traces or transitions between states as part of their input.

Definition 2 (Plan Trace). *A plan trace τ is a finite sequence of states and actions, i.e. $\tau = \langle s_0, a_1, \ldots, a_n, s_n \rangle$. In that, every state s_i has been reached from applying action a_i in state s_{i-1}.*

Trace generation.   Plan traces are intuitively generated in real scenarios by observing an agent interact with an environment. This can easily be done by a software where situations and decisions taken by the agent are recorded. For instance, a domotic house where the housekeeper enter the house and shutdown the alarm, is a clear scenario where states and actions can be observed, and in which "shutdown alarm" action can be learned from observations. Former example shows a trace that is *goal-oriented*, but traces can also be represented by *random-walks* (Amir and Chang 2008) like a robot exploring possible action outcomes in a local scenario which allows to learn action models in an *online* setting. However, we follow the standard *offline* approach in the field, where action models are learned from a given set of traces.

Trace observability.   There are multiple observation levels for every input plan trace τ (Aineto, Celorrio, and Onaindia 2019). We denote Ψ to the function that observes states and Φ to the one that observe actions from each plan trace τ. In full observability environments, $\Psi : S \to S$ and $\Phi : A \to A$ always produce the original state and action respectively. In environments with no observability both functions output $\bot$, which corresponds to no observed action or state. Although, in partial observability both functions could either observe the correct state and action or $\bot$, i.e. $\Psi : S \to S \cup \{\bot\}$ and $\Phi : A \to A \cup \{\bot\}$. The initial state and goal condition (or goal state if its fully defined) are always known for any *goal-oriented* trace. Noise in observations is orthogonal to the observation granularity, in that, other mechanisms can be considered to handle noisy states and actions (Mourao et al. 2012; Zhuo and Kambhampati 2013) in traces, however, we assume noise-free observability.

Input to Λ	$\mathcal{P}$	(connected ?x ?y - place), (agent-at ?x - place), (visited ?x - place)
	$\mathcal{A}'$	(: action **move**(?x ?y - place))
	Σ	p1,p2,...,p63,p64 - place
	$\mathcal{T}$	$\tau_1 = \{s_0, \mathsf{move(p1\ p2)}, \ldots, \mathsf{move(p63\ p64)}, s_n\}, \ldots, \tau_m = \{\ldots\}$
	$\mathcal{F}$	$\sum_{\forall o \in \mathcal{O}} \frac{\text{predicate-errors}(o)}{\text{all-possible-predicates}(o)}$
Output $\mathcal{A}$		Action model for move in Figure 1b

Table 1: Input/Output example for visitall problem

Learning STRIPS Action Models   The task of learning STRIPS action models usually requires more than one plan trace as input. Let us define a set of traces as $\mathcal{T} = \{\tau_i | 1 \leq i \leq m\}$, where m is the number of training samples.

Definition 3 (Learning Action Models Task). *is a tuple $\Lambda = \langle \mathcal{P}, \mathcal{A}', \Sigma, \mathcal{T}, \mathcal{F} \rangle$ where $\mathcal{P}$ is the set of predicates, $\mathcal{A}'$ is a set of partially defined action models, Σ is the set of objects, $\mathcal{T}$ is the set of traces, and $\mathcal{F}$ is the cost function to minimize.*

A solution to Λ is a complete set of action models $\mathcal{A}$, such that $\mathcal{A}' \subseteq \mathcal{A}$. There must be a guarantee that $\mathcal{A}$ minimizes $\mathcal{F}$ in $\mathcal{T}$. The set of predicates $\mathcal{P}$ is used for learning preconditions and effects of action models, while the set of objects Σ is used to validate that $\mathcal{A}$ can correctly generate each planning trace $\tau \in \mathcal{T}$.

The learning process of action models highly depends on the cost function specification $\mathcal{F}$. Often, this is different in every approach, ARMS counts errors over all possible predicates (Yang, Wu, and Jiang 2007), and FAMA uses classical metrics such as precision and recall (Aineto, Celorrio, and Onaindia 2019). Then, for any given cost function metric, two models can be compared to decide which one better represents the set of plan traces. However, when comparing a model with the ground truth (if that exists), only syntactical errors should be taken into account (Aineto, Celorrio, and Onaindia 2019), because semantics do not affect to the dynamics of the environment.

In Table 1 we have an input/output example for the visitall in Figure 1. The input to this problem is the set of predicates introduced in Figure 1b, the set of objects representing the 64 places in the 8×8 grid, and a partial action schema with the move action signature but whose preconditions and effects are empty. We have up to m plan traces that show action sequences for visiting all grid places, and a cost function defined as the total number of errors divided by all possible predicate choices.

STRIPS Action Discovery Task

In this section we introduce our learning task representation for classical planning, explaining the trace properties and the input requirements to guide an unsupervised search

for learning valid action models.

Task representation

The representation of our problem $\Lambda_{r,m}^k$ extends that in Aineto et al. (2018) where semantics of actions (including action signatures) are unknown and the number of parameters for each of the k actions is bounded by a max-arity variable r, and validated over m examples. Action signatures highly restrict the possible preconditions and effects for learning STRIPS action models. We relax that hard constraint in this paper to study the impact of observations for action discovery, in other words, discovering STRIPS action headers and bodies when most of information is lost and only the initial state and objectives of each agent is known.

Traces. Therefore, we observe one or more agents interacting in an environment that can be defined with the same fluents and actions, also known as *classical planning frame* $\mathcal{D} = \langle F, A \rangle$. Agents are represented with a set of m traces $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$ that are *noise-free* and *goal-oriented* in which we only observe their initial and goal states, where the latter might be partially specified. Thus, functions Ψ and Φ always output $\perp$ (no observability), and traces have unknown length. So, we define each trace $\tau_i \in \mathcal{T}$, s.t. $1 \leq i \leq m$, as an initial state s_0^i and a partially defined goal state s_n^i. For simplification, we refer to each trace as the i-th planning instance, i.e. $\tau_i = \langle s_0^i, s_n^i \rangle$ where $s_0^i = I_i$ and $G_i \subseteq s_n^i$. Then, $\mathcal{T} = \{P_1, \ldots, P_m\}$ where each $P_i = \langle \mathcal{D}, I_i, G_i \rangle$.

Predicates and Objects. Even though predicates and objects can be inferred from observed initial states and goal conditions in the traces, we provide $\mathcal{P}$ and Σ as part of the input to ease the compilation to planning. Moreover, we extend the set objects $\Sigma_{r,m}^k$ with k action, r variable and m trace objects, and the set $\mathcal{P}_r^k$ with predicates that represent the arities from 0 to r for each action, precondition, positive and negative effect.

Partial Action Models. The partially defined action models $\mathcal{A}_r^k$ consist of k unnamed actions with a number of parameters that go from 0 to r that is the max-arity value. Then, the total number of partially action models is $|\mathcal{A}_r^k| = k \times (r+1)$. Each $\alpha \in \mathcal{A}_r^k$ is specified with all possible predicate combinations with *well-defined* parameter assignments in the preconditions (Definition 1), and empty effects.

Example 3 (Representation). *Following Example 2, the ground truth shows there is only an action α, $k = 1$, with a max-arity $r = 2$. Then, given $\mathcal{P} = \{\mathsf{at}[\mathsf{x}], \mathsf{v}[\mathsf{x}], \mathsf{conn}[\mathsf{x},\mathsf{y}]\}$, $\Sigma = \{\mathsf{p1}, \mathsf{p2}\}$ and $\tau_1 = \{s_0^1 = \{\mathsf{at}[\mathsf{p1}], \mathsf{v}[\mathsf{p1}], \mathsf{conn}[\mathsf{p1},\mathsf{p2}], \mathsf{conn}[\mathsf{p2},\mathsf{p1}]\}, s_n^1 = \{\mathsf{v}[\mathsf{p1}], \mathsf{v}[\mathsf{p2}]\}\}$. The partially defined action models ($n_\alpha[X]$) are $\alpha_0[]$, $\alpha_1[x]$ and $\alpha_2[x,y]$. Since all predicates have at least one parameter, α_0 has no valid assignments in its preconditions so $\mathsf{pre}_{\alpha_0} = \emptyset$, $\mathsf{pre}_{\alpha_1} = \{\mathsf{at}[\mathsf{x}], \mathsf{v}[\mathsf{x}]\}$, and $\mathsf{pre}_{\alpha_2} = \{\mathsf{at}[\mathsf{x}], \mathsf{at}[\mathsf{y}], \mathsf{v}[\mathsf{x}], \mathsf{v}[\mathsf{y}], \mathsf{conn}[\mathsf{x},\mathsf{y}], \mathsf{conn}[\mathsf{y},\mathsf{x}]\}$, while the effects of all action models are empty $\mathsf{add}_{\alpha_j} = \emptyset$, and $\mathsf{del}_{\alpha_j} = \emptyset$ (for all $0 \leq j \leq r$).*

The reason to remove preconditions (delete predicates) and program effects (add predicates) is to keep action models as restrictive as possible while having a minimum impact in the environment, so that action preconditions are relaxed and effects incremented when more states are observed, until a convergence. This also solves the problem of *learning static predicates* (Gregory and Cresswell 2015), where deciding if they must be included or not in the action preconditions becomes a complex task, so it is much more intuitive and easy to solve when all predicate combinations are included in the preconditions and the decision is about when they should *not* be there.

Cost Function. We intuitively encode the cost function $\mathcal{F}$ as a minimization in the number of editions, where an edition is either removing or programming a predicate in the action model. This can also be reformulated as programming the minimum number of adding ($|\mathsf{add}_\alpha|$) and delete ($|\mathsf{del}_\alpha|$) effects, while removing the minimum number of predicates from preconditions ($-|\mathsf{pre}_\alpha|$), i.e. Equation 1.

$$\mathcal{F} = \frac{1}{|\mathcal{A}'|} \sum_{\alpha \in \mathcal{A}'} (|\mathsf{add}_\alpha| + |\mathsf{del}_\alpha| - |\mathsf{pre}_\alpha|) \qquad (1)$$

The cost function in Equation 1 guides the search to action schemas that could be completely different from the ground truth (if that exists), but this could be still consistent and correct, since complex actions (with many parameters) can be splitted into multiple simpler actions (less parameters) (Areces et al. 2014). Now we are ready to define our *unsupervised learning task* for action discovery as $\Lambda_{r,m}^k = \langle \mathcal{P}_r^k, \mathcal{A}_r^k, \Sigma_{r,m}^k, \mathcal{T}, \mathcal{F} \rangle$, where k is the number of actions in the domain and r is the bound to represent the max-arity for each action. A solution to $\Lambda_{r,m}^k$ is a set of action models $\mathcal{A}$ that subsumes $\mathcal{A}_r^k$ and solve each $\tau \in \mathcal{T}$.

Computing STRIPS Action Models

The computation of our learning task $\Lambda_{r,m}^k$ follows a *compilation-based* approach to PDDL, such that experiments are reproducible and any off-the-shelf classical planner can be used[1]. Firstly, we explain the PDDL representation of each one of the task components.

Objects, Predicates and Fluents The set of objects $\Sigma_{r,m}^k$ is extended with four new types corresponding to variables that bind predicate parameters to action parameters, i.e. Σ_v; copies of the original predicate names Σ_p to relate what is being removed or added in the action model; action objects Σ_a to indentify the action that is being updated or applied; and Σ_t is the set of trace objects. Thus, $\Sigma_{r,m}^k = \Sigma \cup \Sigma_v \cup \Sigma_p \cup \Sigma_a \cup \Sigma_t$ where:

- $\Sigma_v = \{\mathsf{var}_i : 1 \leq i \leq r\}$,
- $\Sigma_p = \{\mathsf{pred}_p : p \in \mathcal{P}\}$,
- $\Sigma_a = \{\mathsf{act}_i : 1 \leq i \leq k\}$,
- $\Sigma_t = \{\mathsf{tr}_i : 1 \leq i \leq m\}$.

---

[1]In case the paper is accepted, we are going to release the framework too.

We describe the following fluents F' as the relation between predicates $\mathcal{P}_r^k$ and objects $\Sigma_{r,m}^k$. There are three kinds of fluents where the first one represents the mode F_{mode}, if it is learning or validating the model; the second F_{ed} consists of removed preconditions and programmed effects from the action schemas; and the last one F_{pred}, represents when a fluent from F holds in the current state of a trace $\tau \in \mathcal{T}$. Then, $F' = F_{\mathsf{mode}} \cup F_{\mathsf{ed}} \cup F_{\mathsf{pred}}$ where :

- $F_{\mathsf{mode}} = \{\mathsf{edit\text{-}mode}, \mathsf{val\text{-}mode}\}$,

- $F_{\mathsf{ed}} = \{\mathsf{rpre}_{p,\sigma}^\alpha, \mathsf{add}_{p,\sigma}^\alpha, \mathsf{del}_{p,\sigma}^\alpha : \alpha \in \Sigma_\alpha, p \in \Sigma_p, \sigma \in \Sigma_v^{\mathsf{ar}(p)}\}$,

- $F_{\mathsf{pred}} = \{\mathsf{holds}_{p,\omega}^\tau : \tau \in \Sigma_t, p \in \Sigma_p, \omega \in \Sigma^{\mathsf{ar}(p)}\}$.

To simplify fluents notation, we refer to action objects as $\alpha \in \Sigma_a$, predicate objects $p \in \Sigma_p$, trace objects $\tau \in \Sigma_t$. Also, we refer to $\omega \in \Sigma^{\mathsf{ar}(p)}$ as a tuple of objects assigned to a predicate p with arity $\mathsf{ar}(p)$, so $\sigma \in \Sigma_v^{\mathsf{ar}(p)}$ is a tuple of $\mathsf{ar}(p)$ variable objects assigned to edition predicates that points to the corresponding action parameters.

Editing and Applying Actions The first task of learning action models $\Lambda_{r,m}^k$ consists of editing the partially defined action models $\mathcal{A}_r^k$ by removing preconditions and programming effects:

$$\mathsf{pre}(\mathsf{edit\text{-}ins}_{p,\sigma}^\alpha) = \{\mathsf{edit\text{-}mode}\} \cup$$
$$\{\mathsf{var}_i \neq \mathsf{var}_j : \mathsf{var}_i \in \sigma, 1 \le i \le \mathsf{ar}(\alpha) < j\},$$
$$\mathsf{ce}(\mathsf{edit\text{-}ins}_{p,\sigma}^\alpha) = \{\emptyset\} \triangleright \{\mathsf{ins}_{p,\sigma}^\alpha\}.$$

The second precondition in edit actions is a constraint that guarantee well-defined action schemas, and word ins has to be substituted either by rpre, add or del predicates. Once the edition of action models finishes, there is an action to start the validation phase:

$$\mathsf{pre}(\mathsf{ed2val}) = \{\mathsf{edit\text{-}mode}\},$$
$$\mathsf{ce}(\mathsf{ed2val}) = \{\emptyset\} \triangleright \{\neg\mathsf{edit\text{-}mode}, \mathsf{val\text{-}mode}\}.$$

The computing continues applying the edited actions until the goal condition is reached for each trace:

$$\mathsf{pre}(\mathsf{apply}_\omega^{\alpha,\tau}) = \{\mathsf{val\text{-}mode}\} \cup$$
$$\{\neg\mathsf{rpre}_{p,\sigma}^\alpha \Rightarrow \mathsf{holds}_{p,\omega}^\tau\}_{\forall p \in \Sigma_p, \sigma \in \Sigma_v^{\mathsf{ar}(p)}},$$
$$\mathsf{ce}(\mathsf{apply}_\omega^{\alpha,\tau}) = \{\{\mathsf{add}_{p,\sigma}^\alpha\} \triangleright \{\mathsf{holds}_{p,\omega}^\tau\} \cup$$
$$\{\mathsf{del}_{p,\sigma}^\alpha\} \triangleright \{\neg\mathsf{holds}_{p,\omega}^\tau\}\}_{\forall p \in \Sigma_p, \sigma \in \Sigma_v^{\mathsf{ar}(p)}}.$$

Example 4 (Variables and objects). *The relation between σ variables and ω objects for each predicate must be well-defined, i.e. in Figure 1b, the positive effect* (visited ?y - place) *is encoded in the action schema as* $\mathsf{add}_{\mathsf{visited},\langle\mathsf{var}_2\rangle}^{\mathsf{move}}$, *so applying a move action between positions* p1 *and* p2, *grounds the action signature to* $\mathsf{apply}_{\langle\mathsf{p1},\mathsf{p2}\rangle}^{\mathsf{move},\mathsf{tr}_1}$, *and the positive effect will be* $\mathsf{holds}_{\mathsf{visited},\langle\mathsf{p2}\rangle}^{\mathsf{tr}_1}$ *which means* p2 *has been visited in the first trace.*

DAM Compilation We named our compilation **D**iscovering **A**ction **M**odels (DAM). It is a compilation to a classical planning problem $P' = \langle F', A', I', G' \rangle$ where the set of fluents is $F' = F_{\mathsf{mode}} \cup F_{\mathsf{ed}} \cup F_{\mathsf{pred}}$. A set of actions for each one of the three phases: the edition phase $A_{\mathsf{ed}} = \{\mathsf{edit\text{-}ins}_{p,\omega}^\alpha : \alpha \in \Sigma_a, p \in \Sigma_p, \sigma \in \Sigma_v^{\mathsf{ar}(p)}\}$; the transition from edition to validation $A_{\mathsf{e2v}} = \{\mathsf{ed2val}\}$; and the trace validation phase with applying actions $A_{\mathsf{app}} = \{\mathsf{apply}_\omega^{\alpha,\tau} : \alpha \in \Sigma_a, \tau \in \Sigma_t, \omega \in \Sigma^{\mathsf{ar}(\alpha)}\}$. Thus, $A' = A_{\mathsf{ed}} \cup A_{\mathsf{e2v}} \cup A_{\mathsf{app}}$. The new initial state and goal condition are defined as the set of all fluents that hold in the initial state and the goal condition of every trace, i.e. $I' = \{\mathsf{holds}_{p,\omega}^i : p[\omega] \in I_i, 1 \le i \le m\}$, and $G' = \{\mathsf{holds}_{p,\omega}^i : p[\omega] \in G_i, 1 \le i \le m\}$.

Theoretical properties

In this section we prove the theoretical properties of soundness and completeness of our approach given the number of actions k, the max-arity r and the set of m traces.

Theorem 1 (Soundness). *Any classical plan π that solves P' induces a set of action models $\mathcal{A}$ that solves $\Lambda_{r,m}^k = \langle \mathcal{P}_r^k, \mathcal{A}_r^k, \Sigma_{r,m}^k, \mathcal{T}, \mathcal{F} \rangle$.*

Proof sketch. The first actions in π are A_{ed} because of edit-mode fluent, in that, preconditions are removed and effects are programmed for each partially defined action model $\mathcal{A}_r^k$. The next is an action A_{e2v} which finishes editing actions and starts the validation phase. Once in validation, a sequence of edited actions from A_{app} is computed for each trace $\tau \in \mathcal{T}$ in parallel, mapping all initial states I_i to goal conditions G_i s.t. $1 \le i \le m$. Thus, all traces are solved by the set of edited action models $\mathcal{A}$ induced from $A_{\mathsf{ed}} \cap \pi$, which is the definition of $\mathcal{A}$ solving $\Lambda_{r,m}^k$. $\square$

Theorem 2 (Completeness). *Any set of action models $\mathcal{A}$ that solves $\Lambda_{r,m}^k$ can compute a plan π that solves $P' = \langle F', A', I', G' \rangle$.*

Proof sketch. Action models $\mathcal{A}$ do not require any edition since they are already a solution of the learning task $\Lambda_{r,m}^k$. Moreover, a sequence of grounded $\mathcal{A}$, named π_Λ, solve all traces $\tau \in \mathcal{T}$ by definition. Thus, each action in π_Λ can be compiled to an action in A_{app}, which applied in the same sequence in π map each initial state I_i to its goal condition G_i s.t. $1 \le i \le m$, in other words π maps the initial state I' to a goal condition G', which encode the initial states and goal conditions of all traces, solving P'. $\square$

Extension to Partial Observability

The inputs for learning action models usually have some level of observability, and not the extreme view we assume where only initial states and goal conditions are observed. The DAM compilation can easily be extended by adding *validating actions* such as (Aineto, Celorrio, and Onaindia 2019; Aineto, Jiménez, and Onaindia 2018), that specify the order in which intermediate states must hold, and a goal condition extended with a new fluent that assess when intermediate states have been correctly validated. However, the

problem can be simplified to vanilla DAM, where every pair of consecutive and fully defined states in the traces can be encoded as new planning problems of initial and goal states. For instance, a single trace $\tau = \langle s_0, s_1, s_2 \rangle$ can be splitted into $\tau_1 = \langle s_0, s_1 \rangle$ and $\tau_2 = \langle s_1, s_2 \rangle$ which is the input to standard DAM compilation.

Unsupervised Discovering Action Models

The DAM compilation for computing STRIPS action models assumes the number of action schemas and max-arity bounds are given. UDAM navigates through the space of configurations and invokes repeatadly the DAM compilation. The overall process is shown in Algorithm 1. The phases are as follows:

Algorithm 1 Unsupervised Discovering Action Models (UDAM)

   Input: $\mathcal{P}$ predicates, Σ objects and $\mathcal{T}$ traces
   Output: $\mathcal{A}$ action models
1: $\pi^* \leftarrow \emptyset$
2: $k, r, m \leftarrow 1, 0, |\mathcal{T}|$
3: $P' \leftarrow \mathsf{compile}(\Lambda_{r,m}^k)$
4: $open \leftarrow \{\langle P', k, r \rangle\}$
5: $close \leftarrow \emptyset$
6: **while** $open \neq \emptyset$ **do**
7:    $\langle P', k, r \rangle \leftarrow argmin_{\langle P', k, r \rangle \in open} |\mathsf{operators}(P')|$
8:    $open \leftarrow open \setminus \{\langle P', k, r \rangle\}$
9:    $close \leftarrow close \cup \{\langle P', k, r \rangle\}$
10:    $\pi \leftarrow \mathsf{madagascar}(P')$
11:    **if** $\mathsf{sat}(\pi)$ **and** $(\mathsf{cost}(\pi) < \mathsf{cost}(\pi^*)$ **or** $\pi^* = \emptyset)$ **then**
12:       $\pi^* \leftarrow \pi$
13:    **end if**
14:    $k' \leftarrow k + 1$
15:    $P_1' \leftarrow \mathsf{compile}(\Lambda_{r,m}^{k'})$
16:    **if** $k' \leq 2 \times |\mathcal{P}|$ **and** $\langle P_1', k', r \rangle \notin \{open \cup close\}$ **then**
17:       $open \leftarrow open \cup \{\langle P_1', k', r \rangle\}$
18:    **end if**
19:    $r' \leftarrow r + 1$
20:    $P_2' \leftarrow \mathsf{compile}(\Lambda_{r',m}^k)$
21:    **if** $r' \leq |\Sigma|$ **and** $\langle P_2', k, r' \rangle \notin \{open \cup close\}$ **then**
22:       $open \leftarrow open \cup \{\langle P_2', k, r' \rangle\}$
23:    **end if**
24: **end while**
25: $\mathcal{A} \leftarrow \mathsf{induce}(\pi^*)$
26: **return** $\mathcal{A}$

Phase 1 - Initializing. Lines 1 to 5 initialize the variables, where the best plan π^* is empty, there is one action $k = 1$ without arity $r = 0$, the number of traces is fixed $m = |\mathcal{T}|$, and a new planning problem is generated with $\mathsf{compile}(\Lambda_{r,m}^k)$ which is defined in the DAM Compilation Section. Then, there are open and close lists, to denote visited and expanded configuration nodes, where each configuration is a parameter assignment to the learning task.

Phase 2 - Searching. While there are configurations to explore in the open list, the one with lowest number of grounded operators is selected, i.e. $|\mathsf{operators}(P')|$. Therefore, the selected configuration is removed from the open list and added to the close list. Then, a plan π is computed for problem P' using a SAT-based planner called Madagascar (Rintanen 2014). If π is satisfiable, i.e. $\mathsf{sat}(\pi)$, and its cost defined in Equation 1 is lower than the cost of the best plan π^* or if it is empty, then there is a new best plan.

Phase 3 - Expanding. Two new compilations are added to the open list, if they have not been previously explored. The first increments by one the number of action signatures, while the second increments by one the max-arity. Actions are theoretically bounded in the number of predicates mutiplied by 2, which means actions to add and delete each predicate. Arity is bounded in the number of objects, which means an object for each parameter of an action. It continues in Phase 2 until the open list is empty.

Phase 4 - Inducing. Once all available configurations have been explored, the set of action models $\mathcal{A}$ is induced from the best plan π^* following the proof in Theorem 1.

Other strategies can be applied to Algorithm 1 such as search for first configuration that is satisfiable, or use a different function to select the next tuple from the open list. These changes will might induce different sets of action models but they will still be sound and complete, since they only change the order in which the space of configurations is explored.

Experiments

The two main evaluations for learning action models consist of their computation and posterior validation on new problems. Even though, UDAM algorithm can compute STRIPS action models from predicates, objects and traces, it requires a human interpretation if we want to explain the generated model and check for a valid model generalization. Previous methods for model learning use existing planning models from the International Planning Competitions (IPCs) (López, Celorrio, and Olaya 2015). In that domains the dynamics are known, so they become the perfect benchmark to test learnt action similarity, or even how different number of actions and parameters relate to the original domain. The intuitive outcome from this, is that real domains can be learnt with UDAM if the learnt models are similar to expert models.

Settings. All experiments are evaluated using a SAT-based planner called Madagascar (Rintanen 2014), a memory bound of 8GB and a time-out for each instance is set to 30 minutes in an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz laptop.

Input. The domains and problems from IPCs are conveniently gathered in the PLANNING.DOMAINS repository (Muise 2016). Selected domains are: *hanoi*, where a

Table 2: Results for classical problem of the IPC. *Alg. total user time* is the total CPU time taken by the algorithm. *User time first sol.*, *Memory first sol. (MB)* are, respectively the CPU time spent and the peak memory usage for finding the first action model. *User time best sol. (s)* and *Memory best sol. (MB)* are the CPU time and peak memory usage for finding the model with the least cost. *Lowest cost* is the minimum cost among the found action models. *Coverage best sol.* is the fraction of validation instances that have been solved with the best model found.

Domain	Alg. total user time (s)	User time first sol. (s)	Memory first sol. (MB)	User time best sol. (s)	Memory best sol. (MB)	Lowest cost	Coverage best sol.
hanoi	1580.43	1411.02	41.71	1485.90	301.18	-1.50	30/30
blocks	344.44	114.18	17.33	319.95	127.97	-3.14	30/30
visitall	283.47.08	150.69	14.88	199.56	32.44	-0.80	30/30

certain amount of disks must be placed from leftmost peg to rightmost one; *blocks*, where towers of blocks in a given setting have to be arranged in a different order, and *visitall* where an agent must visit all cells in a grid. In case of no observability, we only need the planning instances, but with partial observability, we can extract partial sequences of states by running a planner such as Fast Downward (Helmert 2006), that can be configured to find either a satisfying or an optimal solution. The set of predicates and objects are known from planning instances. We have fed our algorithm with 5 traces from each domain.

Computing action models

We measure the performance of the computation in terms of time and memory usage. These results are reported in Table 2. The reported figures are averaged over 6 executions of the algorithm, each time with 5 traces. An interesting phenomena that can be noted is that UDAM spends most of the time finding the first solution. Once the first model is found, further models are found much more quickly. The reason is that Madagascar takes much more time to ascertain that a configuration is not viable than to find a solution for a viable configuration. Most of the configurations tried at the beginning by UDAM do not have solution, so a large amount of time is spent to rule out these configurations.

The first solution found is, typically, one whose preconditions are excessively relaxed. For instance, in the case of the *hanoi* domain, the first solution that is found is one with two actions with arity two. One of these actions resembles a *unstack* operator from the *blocksworld* domain, while the other one resembles a *stack* operator. However, *hanoi* does not define any predicate to indicate that a disk is grasped, so there is no way to ensure that the *unstack*-like operator will be applied to a disk that is not currenlty stacked on top of another one.

Since UDAM prefers models with a larger number of preconditions in comparison to the number of effects, it will retain models in which the natural actions that an human expert would suggest are broken into several smaller actions that average a larger number of preconditions. This could represent an issue since these models allow transitions that would not be allowed by the expert's domain.

Validation

Precision and recall have been typical metrics to analyze machine learning models. Recently, they have been adapted to compute action models (Aineto, Jiménez, and Onaindia 2018). One of the main limitations to consider this metrics in planning, is that ground truth models are required to know how far are solutions from human expert models. In addition, when the action signatures are not known beforehand, there is no baseline to compare, increasing the difficulty of action interpretation.

In our approach we assume that ground truth model is unknown, if that exists. Thus, the best model is computed by minimizing the cost function defined in Equation 1. Notice that this cost can be negative if, in average, there are more preconditions than effects.

To validate, we compute the *coverage* over a set of validation instances different from the one used by the algorithm to infer the models. A total of 30 instances have been used to compute the coverage. We see that, even without the action signatures, our algorithm is able to find an action model that can solve all the validation instances. While this means that the found models can solve the instances they are meant to solve, they potentially allow transitions that are not allowed by the expert's domains. We are currently seeking ways to mitigate this shortcoming.

Related Work

Learning action models has been a topic of interest for long time (Gil 1994; Benson 1995; Wang 1995). In those, the process of generating knowledge was incremental, and planning operators were refined by observing traces of an agent interacting in complex environments. Then, ARMS algorithm (Yang, Wu, and Jiang 2007) was capable to generate deterministic planning operators, while SLAF (Amir and Chang 2008) was learning partially observable operators from predicates, objects and sequences of actions. Recent work in learning action models (Aineto, Celorrio, and Onaindia 2019; Aineto, Jiménez, and Onaindia 2018) were capable of computing STRIPS operators from only initial and goal states using a classical planner. We have built our compilation based on the latter, where not only preconditions and effects must be learnt but action signatures too. Also, we solve the problem of programming static predicates by initializing partial action schemas with all possible predicate-parameter combinations and applying actions that remove

preconditions. In addition, we propose an unsupervised algorithm (UDAM) capable of learning action models without any bound specification, just using theoretical bounds.

In contrast to *system-centric* approaches (Arora et al. 2018), there is the LOCM family of algorithms (Cresswell, McCluskey, and West 2009; Cresswell and Gregory 2011; Gregory and Cresswell 2015). In those, the inputs are full observable traces of actions without any state or predicate information, producing an *object-centric* domain represented with finite state machines. However, planning actions usually interact with multiple objects at the same time, and depending on the level of trace observation it affects to learn the correct transition system as it is highly dependant on observed object types.

There have been further research that require statistical analysis because inputs are noisy. For instance, when the actions observed in plan traces could be wrong observations (Zhuo and Kambhampati 2013), or when states in plan traces are partially observable and noisy (Mourao et al. 2012). Also, model learning can be applied to more expressive solutions beyond primitive actions, i.e. learning hierarchical task networks (Zhuo, Muñoz-Avila, and Yang 2014), generating planning operators *on-demand* with a model-free approach (Martínez, Alenya, and Torras 2017), or the CAMA method for learning the models from human annotators (crowdsourcing) and plan traces (Zhuo 2015). All these approaches are orthogonal to UDAM in that they can be applied to solve the partial/noise observability in traces and guiding cost functions from crowd knowledge.

Last but not least, most work in learning action models, like ARMS or FAMA, assume all action signatures are observable in the plan traces or in a partial model of action schemas. Although, there is some work that explores to learn models from non-symbolic inputs to plan in the latent space such as AMA2 (Asai and Fukunaga 2018) and LatPlan for goal recognition (Amado et al. 2018). Both require all transitions as part of the input and only the latter generates PDDL action schemas (without guarantees) from the bits that change in the latent space. In our case, the learning task is similar to AMA2 or LatPlan, in the sense that has no information about actions, and akin to FAMA because intermediate states are non-observable. Thus, to the best of our knowledge, UDAM is the first unsupervised approach to generate STRIPS action models from non-observable traces, where only initial states and goal conditions are known.

Conclusions

In this paper we have explored the unsupervised generation of action models, where not only preconditions and effects are learnt but action signatures too. We have followed a *compilation-based* approach to classical planning, so that we can use any off-the-shelf classical planner. We propose a new method in the DAM compilation, that mitigates the problem of learning static predicates, in that preconditions are removed instead of programmed (added) as in previous approaches. In the experiments, we use the IPC benchmarks to evaluate the convergence of our approach for model learning, the performance in the computation and the generalization when validating learnt action models compared to previous methods. Our validation shows that instances that are meant to be solvable are, effectively, solved by our algorithm. However, preconditions may be too weak.

As future work we consider to introduce negative examples to improve the model quality with a more informed cost function, akin to *Inductive Logic Programming* (Muggleton and De Raedt 1994). Also, we want to address the connection between computing minimal action schemas and *least commitment planning* (Weld 1994), the generation of simpler actions (less parameters) from complex actions (with many parameters) (Areces et al. 2014), and proving that the optimal set of action models (cost function based) is as complex as finding an optimal plan (Helmert, Röger, and others 2008). In addition, we want to apply UDAM algorithm to robotics, where a human can guide a robot to learn high-level skills from sensor information. Thus, our interests rely in a valid framework that can unsupervisedly compute with guarantees a high-level model from non-symbolic data such as the following methods (Asai and Fukunaga 2018; Amado et al. 2018; Bonet and Geffner 2019).

Acknowledgments

References

[Aineto, Celorrio, and Onaindia 2019] Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence* 275:104–137.

[Aineto, Jiménez, and Onaindia 2018] Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning strips action models with classical planning. In *International Conference on Automated Planning and Scheduling, ICAPS-18.*

[Amado et al. 2018] Amado, L.; Pereira, R. F.; Aires, J.; Magnaguagno, M.; Granada, R.; and Meneguzzi, F. 2018. Goal recognition in latent space. In *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE.

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

[Areces et al. 2014] Areces, C.; Bustos, F.; Dominguez, M. A.; and Hoffmann, J. 2014. Optimizing planning domains by automatic action schema splitting. In *ICAPS.*

[Arora et al. 2018] Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A review of learning planning action models. *The Knowledge Engineering Review* 33.

[Asai and Fukunaga 2018] Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Thirty-Second AAAI Conference on Artificial Intelligence.*

[Benson 1995] Benson, S. 1995. Inductive learning of reactive action models. In *Machine Learning Proceedings 1995*. Elsevier. 47–54.

[Blythe, Deelman, and Gil 2004] Blythe, J.; Deelman, E.; and Gil, Y. 2004. Automatically composed workflows for grid environments. *IEEE Intelligent Systems* 19(4):16–23.

[Bonet and Geffner 2019] Bonet, B., and Geffner, H. 2019. Learning first-order symbolic planning representations from plain graphs. *arXiv preprint arXiv:1909.05546*.

[Carman, Serafini, and Traverso 2003] Carman, M.; Serafini, L.; and Traverso, P. 2003. Web service composition as planning. In *ICAPS 2003 workshop on planning for web services*, 1636–1642.

[Cresswell and Gregory 2011] Cresswell, S., and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *ICAPS*.

[Cresswell, McCluskey, and West 2009] Cresswell, S.; McCluskey, T. L.; and West, M. M. 2009. Acquisition of object-centred domain models from planning examples. In *ICAPS*.

[Fikes and Nilsson 1971] Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4):189–208.

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

[Gil 1994] Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Machine Learning Proceedings 1994*. Elsevier. 87–95.

[Gnad et al. 2019] Gnad, D.; Torralba, A.; Domínguez, M.; Areces, C.; and Bustos, F. 2019. Learning how to ground a plan–partial grounding in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7602–7609.

[Gregory and Cresswell 2015] Gregory, P., and Cresswell, S. 2015. Domain model acquisition in the presence of static relations in the lop system. In *ICAPS*, 97–105.

[Helmert, Röger, and others 2008] Helmert, M.; Röger, G.; et al. 2008. How good is almost perfect?. In *AAAI*, volume 8, 944–949.

[Helmert 2006] Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

[Kambhampati 2007] 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*, volume 22, 1601. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

[López, Celorrio, and Olaya 2015] López, C. L.; Celorrio, S. J.; and Olaya, Á. G. 2015. The deterministic part of the seventh international planning competition. *Artificial Intelligence* 223:82–119.

[Martínez, Alenya, and Torras 2017] Martínez, D.; Alenya, G.; and Torras, C. 2017. Relational reinforcement learning with guided demonstrations. *Artificial Intelligence* 247:295–312.

[Mourao et al. 2012] Mourao, K.; Zettlemoyer, L. S.; Petrick, R.; and Steedman, M. 2012. Learning strips operators from noisy and incomplete observations. *arXiv preprint arXiv:1210.4889*.

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

[Muise 2016] Muise, C. 2016. Planning.Domains. In *ICAPS - Demonstrations*.

[Rintanen 2014] Rintanen, J. 2014. Madagascar: Scalable planning with sat. *Proceedings of the 8th International Planning Competition (IPC-2014)* 21.

[Segovia-Aguas, Jiménez, and Jonsson 2017] Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2017. Unsupervised classification of planning instances. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017); 2017 June 18-23; Pittsburgh, USA. Palo Alto, CA: AAAI; 2017. p. 452-60.* Association for the Advancement of Artificial Intelligence (AAAI).

[Wang 1995] Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Machine Learning Proceedings 1995*. Elsevier. 549–557.

[Weld 1994] Weld, D. S. 1994. An introduction to least commitment planning. *AI magazine* 15(4):27.

[Yang, Wu, and Jiang 2007] 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.

[Zhuo and Kambhampati 2013] Zhuo, H. H., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *IJCAI*, 2444–2450.

[Zhuo, Muñoz-Avila, and Yang 2014] Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2014. Learning hierarchical task network domains from partially observed plan traces. *Artificial intelligence* 212:134–157.

[Zhuo 2015] Zhuo, H. H. 2015. Crowdsourced action-model acquisition for planning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.