Online Learning of Action Models for PDDL Planning

Leonardo Lamanna[1,2] , **Alessandro Saetti**[1] , **Luciano Serafini**[2] ,
Alfonso E. Gerevini[1] and **Paolo Traverso**[2]

[1]Department of Information Engineering, University of Brescia, Italy
[2]Fondazione Bruno Kessler (FBK), Trento, Italy
{l.lamanna, alessandro.saetti, alfonso.gerevini}@unibs.it, {llamanna, serafini, traverso}@fbk.eu

Abstract

The automated learning of action models is widely recognised as a key and compelling challenge to address the difficulties of the manual specification of planning domains. Most state-of-the-art methods perform this learning offline from an input set of plan traces generated by the execution of (successful) plans. However, how to generate informative plan traces for learning action models is still an open issue. Moreover, plan traces might not be available for a new environment. In this paper, we propose an algorithm for learning action models *online*, incrementally during the execution of plans. Such plans are generated to achieve goals that the algorithm decides online in order to obtain informative plan traces and reach states from which useful information can be learned. We show some fundamental theoretical properties of the algorithm, and we experimentally evaluate the online learning of the actions models over a large set of IPC domains.

1 Introduction

Automated planning techniques require the specification of planning domains through action models (a set of preconditions and a set of effects for each domain action). However, the manual specification of the action models is often an inaccurate, time consuming, and error-prone task. The automated learning of action models is widely recognised as a key and compelling challenge to overcome these difficulties. Several works have addressed the task of learning action models, and have provided important results from different perspectives and according to different assumptions, see, e.g., [Yang *et al.*, 2007; Amir and Chang, 2008; Xu and Laird, 2010; Rodrigues *et al.*, 2011; Mourão *et al.*, 2012; Zhuo and Kambhampati, 2013; Cresswell *et al.*, 2013; Certicky, 2014; Aineto *et al.*, 2018; Aineto *et al.*, 2019].

However, most of the recent and state-of-the-art methods perform learning offline, and require as input a set of plan traces generated by previously executed actions. This has two major drawbacks. First, often agents need to learn the model of the domain *online*, because they need to explore an unknown environment, acquire information, and learn a model by experimenting the execution of their actions incrementally, step by step. This is the case of many applications in robotics, e.g., in SLAM [Stachniss *et al.*, 2016], where the robot tries to build a map of the environment by exploration, or in the Robocup Rescue [Kitano and Tadokoro, 2001], where the robot needs to explore the environment to perform a rescue task. Second, previous work on learning action models does not deal with the problem of generating informative plan traces. As stated in the conclusions of [Aineto *et al.*, 2019], generating informative plan traces for learning planning action models is still an open issue. Indeed, if the available set of plan traces does not contain informative examples, there is little chance to learn all action preconditions, since some preconditions can be only discovered by specific plans that can unlikely be generated randomly [Fern *et al.*, 2004].

In this paper, we propose a new approach that does not suffer from these drawbacks, focusing on the case of learning STRIPS action schema expressed in PDDL, and under the assumption of full observability of the states reached by the agent. We propose an algorithm, called OLAM algorithm (Online Learning of Action Models), for learning action models online, incrementally during the execution of plans. A key aspect of OLAM is that it combines and interleaves the activity of learning action preconditions and effects with an exploration phase that selects which plan to execute. In this way, OLAM generates plan traces to reach certain goal states, decided online, which are useful for the learning task.

Beyond proving termination, we analyse our algorithm to show some important theoretical properties that are defined according to the state transitions of the models learned by the algorithm. In particular we prove that OLAM is correct, i.e., it learns action models which generate only the state transitions generated by the planning domain modelling the true environment where the agent acts. Moreover OLAM is "integral", i.e., it learns action models that generate all the transitions of the true environment with respect to the states that can be reached by the algorithm.

We also provide substantial empirical evidence of the good learning performance of OLAM using a large set of benchmarks from the International Planning Competitions (IPCs). Finally we experimentally compare OLAM with a recent and state-of-the-art method for learning action models offline, showing that the online learning can be much more effective.

2 Related Work

Recent offline approaches address the problem of model learning with different assumptions on the observability of states and actions, see, e.g., [Amir and Chang, 2008; Bonet and Geffner, 2020; Cresswell *et al.*, 2013; Mourão *et al.*, 2012; Newton *et al.*, 2007; Yang *et al.*, 2007; Zhuo *et al.*, 2010; Zhuo and Kambhampati, 2013]. A prominent system among these is Fama [Aineto *et al.*, 2019], which learns action models offline from examples by transforming the learning task into a classical planning task. It works with different kinds of inputs, from a set of plans to just a pair of initial and final states, without intermediate actions or states. Moreover, it accepts in input partially specified action models.

On the one hand, the aforementioned approaches to offline learning can deal with partial observability of states and actions, and some of them even with noisy states and noisy actions. OLAM requires instead full observability of states, it does not deal with noisy sensors, and actions are decided by OLAM itself. On the other hand, differently from OLAM, all these approaches are offline, require in input plan traces that in some cases might be not available, and hence do not deal with the issue of selecting informative plan traces.

Since the seminal work on online learning of operators [Gil, 1994b; Gil, 1994a; Wang, 1996], and the first approaches to learning action models by integrating learning, planning, and execution [García-Martínez and Borrajo, 2000], some recent approaches have addressed the problem of online and incremental learning of action models. Walsh and Littman (2008) propose an approach to online learn action models which can be used in web-service planning problems. Their approach requires the use of an external "teacher" providing plan traces on demand. 3SG [Certicky, 2014] is an online algorithm that learns probabilistic action models with conditional effects and deals with action failures, sensory noise, and incomplete information. Xu and Laird (2010) describes an instance-based online method for learning action models in relational domains. The work is extended to deal with both discrete and continuous action models [Xu and Laird, 2011; Xu and Laird, 2013]. Rodrigues *et al.* (2010a; 2010b) propose a technique based on relational reinforcement learning to learn deterministic action models, and Rodrigues *et al.* (2011) extend the approach to deal with nondeterministic actions. These approaches are based on important technical differences with respect to our work. Most important, the main conceptual and practical difference is that all these approaches assume that the action to be executed is randomly selected or given in input, and therefore do not deal with the problem of guiding the exploration phase towards informative states, a key and promising feature of OLAM. The work by Lamanna *et al.* (2021) proposes an online method to learn planning domains by mapping continuous observations to deterministic state transition systems. It uses a given PDDL planning domain and a classical planner to heuristically explore the state space towards the problem goal. However, it focuses on learning the final state machine of the planning domain rather than PDDL action models like OLAM.

Our approach shares some similarities with the work on planning by reinforcement learning (RL) [Sutton and Barto, 1998]. However, RL focuses on learning policies rather than PDDL action models. Moreover, most often, in RL, actions are represented as (probabilistic) state transitions, rather than with symbolic action models.

3 Problem

Let $\mathcal{P}$ be a set of predicates with associated arity, of a first order language, and $\mathcal{O}$ be a finite set of operator names with associated arity. Predicates and operators of arity n are called n-ary predicates and n-ary operators. Given an n-tuple $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ of distinct symbols (constants or variables), let $\mathcal{P}(\boldsymbol{x})$ be the set of atomic formulas $p(x_{i_1}, \ldots, x_{i_m})$ obtained by applying the m-ary predicate $p \in \mathcal{P}$ to any m-tuple of symbols $\langle x_{i_1}, \ldots, x_{i_m} \rangle$ in $\boldsymbol{x}$ (with $1 \leq i_1, \ldots, i_m \leq n$). For instance, if $\mathcal{P}$ contains the single binary predicate on, and $\boldsymbol{x} = \langle x_1, x_2, x_3 \rangle$. Then, $\mathcal{P}(\boldsymbol{x}) = \{\texttt{on}(x_i, x_j) \,|\, 1 \leq i, j \leq 3\}$.

Definition 1 (Action schema)**.** *An* action schema *for an n-ary operator name $op \in \mathcal{O}$ on the set of predicates $\mathcal{P}$ is a tuple $\langle \mathsf{par}(op), \mathsf{pre}(op), \mathsf{eff}^+(op), \mathsf{eff}^-(op) \rangle$, where $\mathsf{par}(op)$ is a tuple of variables, $\mathsf{pre}(op)$, $\mathsf{eff}^+(op)$, and $\mathsf{eff}^-(op)$ are three sets of atoms on $\mathcal{P}(\mathsf{par}(op))$.*

Essentially, $\mathsf{pre}(op)$, $\mathsf{eff}^+(op)$, and $\mathsf{eff}^-(op)$ represent the preconditions, positive, and negative effects of operator op. Without loss of generality, we assume that operators have no negative precondition. We also assume that the description of the effects is consistent, i.e., $\mathsf{eff}^+(op) \cap \mathsf{eff}^-(op) = \emptyset$.

Definition 2 (Ground action)**.** *The* ground action $a = op(c_1, \ldots, c_n)$ *of an n-ary operator name $op \in \mathcal{O}$ w.r.t. the constants $c_1, \ldots, c_n$ is the triple $\langle \mathsf{pre}(a), \mathsf{eff}^+(a), \mathsf{eff}^-(a) \rangle$, where $\mathsf{pre}(a)$ (resp. $\mathsf{eff}^+(a)$, $\mathsf{eff}^-(a)$) is obtained by replacing the i-th parameter of $\mathsf{par}(op)$ in $\mathsf{pre}(op)$ (resp. $\mathsf{eff}^+(op)$, $\mathsf{eff}^-(op)$) with c_i.*

We use the term *lifted*, as the opposite of *grounded*, to refer to expressions and actions where constants have been replaced with parameters.

Definition 3 (Planning domain)**.** *A planning domain $\mathcal{M}$ is a triple $\langle \mathcal{P}, \mathcal{O}, \mathcal{H} \rangle$ where $\mathcal{P}$ is a set of predicates, $\mathcal{O}$ is a set of operator names with their arity and, for every $op \in \mathcal{O}$, $\mathcal{H}$ is a function mapping an operator name op into an action schema.*

Definition 4 (Finite-State Machine of a planning domain)**.** *The* Finite-State Machine (FSM) *of a planning domain $\mathcal{M} = \langle P, \mathcal{O}, \mathcal{H} \rangle$ for the set $\mathcal{C}$ of constants is the triple $\mathcal{M}(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta \rangle$ where $\mathcal{S} = 2^{\mathcal{P}(\mathcal{C})}$ is the set of all possible subsets of facts; $\mathcal{A}$ is the set of all possible ground actions of each n-ary operator name in $\mathcal{O}$ on any n-tuple of constants in $\mathcal{C}$; $\delta \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is a transition relation such that $(s, a, s') \in \delta$ if $\mathsf{pre}(a) \subseteq s$ and $s' = s \cup \mathsf{eff}^+(a) \setminus \mathsf{eff}^-(a)$.*

A *plan* π in $\mathcal{M}(\mathcal{C})$ is a finite sequence of actions. A state $s_n \in S$ is *reachable* from a state $s_0 \in S$ in $\mathcal{M}(\mathcal{C})$ if there is a plan $\pi = \langle a_1, \ldots, a_n \rangle$ such that $(s_{i-1}, a_i, s_i) \in \delta$ for $i = 1 \ldots n$.

Assuming that the sets $\mathcal{P}$, $\mathcal{O}$ and $\mathcal{C}$ are known by the agent, its task is to *learn a planning domain by executing the actions available in $\mathcal{O}$ over constants in $\mathcal{C}$, observing,*

and determining what are their preconditions and effects on the environment described in terms of the properties in $\mathcal{P}$. In formal terms, the agent has to build an action model $\mathcal{M} = \langle \mathcal{P}, \mathcal{O}, \mathcal{H} \rangle$, i.e., the preconditions and effects of every action schema in the domain of $\mathcal{H}$. We assume that the dynamics of the environment where the agent acts, which is unknown by the agent, is fully described by the finite state machine $\mathcal{M}'(\mathcal{C})$, where $\mathcal{M}' = \langle \mathcal{P}, \mathcal{O}, \mathcal{H}' \rangle$ is an action model called *Ground-Truth Model* (GTM).

The following definitions state the notions of correctness and integrity for the learned planning domain $\mathcal{M} = \langle P, \mathcal{O}, \mathcal{H} \rangle$ w.r.t. the GTM.

Definition 5 (Correctness)**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two action models and $\mathcal{M}(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta \rangle$ and $\mathcal{M}'(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta' \rangle$ be their FSMs with respect to a set of constants $\mathcal{C}$. We say that*

1. *$\mathcal{M}(\mathcal{C})$ correctly approximates $\mathcal{M}'(\mathcal{C})$ from a state $s_0 \in \mathcal{S}$ if, for every state s_n reachable from s_0 in $\mathcal{M}(\mathcal{C})$, $\langle s_n, a, s \rangle \in \delta$ implies $\langle s_n, a, s' \rangle \in \delta'$ for some $s' \supseteq s$.*
2. *$\mathcal{M}(\mathcal{C})$ correctly approximates $\mathcal{M}'(\mathcal{C})$ if $\mathcal{M}(\mathcal{C})$ correctly approximates $\mathcal{M}'(\mathcal{C})$ from every state in $\mathcal{S}$;*
3. *$\mathcal{M}$ correctly approximates $\mathcal{M}'$ if $\mathcal{M}(\mathcal{C})$ correctly approximates $\mathcal{M}'(\mathcal{C})$ for every set of constants $\mathcal{C}$.*

A plan is *valid* when the actions in the plan are "executable" and the plan achieves a given set of (positive) goals. Therefore, when the learned model correctly approximates the GTM, any valid plan computed by using the learned model is also valid for the GTM.

Definition 6 (Integrity)**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two action models and $\mathcal{M}(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta \rangle$ and $\mathcal{M}'(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta' \rangle$ be their FSMs with respect to a set of constants $\mathcal{C}$. We say that*

1. *$\mathcal{M}(\mathcal{C})$ integrally approximates $\mathcal{M}'(\mathcal{C})$ from a state $s_0 \in \mathcal{S}$ if, for every state s_n reachable from s_0 in $\mathcal{M}(\mathcal{C})$, $\langle s_n, a, s' \rangle \in \delta'$ implies $\langle s_n, a, s \rangle \in \delta$ for some $s \supseteq s'$;*
2. *$\mathcal{M}(\mathcal{C})$ integrally approximates $\mathcal{M}'(\mathcal{C})$ if $\mathcal{M}(\mathcal{C})$ integrally approximates $\mathcal{M}'(\mathcal{C})$ from every state in $\mathcal{S}$;*
3. *$\mathcal{M}$ integrally approximates $\mathcal{M}'$ if $\mathcal{M}(\mathcal{C})$ integrally approximates $\mathcal{M}'(\mathcal{C})$ for every set of constants $\mathcal{C}$.*

Therefore, when the learned model integrally approximates the GTM, any valid plan for the GTM is also a valid plan for the learned model.

4 Learning Algorithm

In the proposed approach, the agent constructs and executes informative plan traces for learning the planning domain. Algorithm 1 shows the pseudocode of the OLAM (*Online Learning of Action Models*). The input of the algorithm are the same sets of predicates and operator names (with their associated arity) of the GTM, and a set $\mathcal{C}$ of constants representing the objects of the environment explored by the agent. OLAM produces in the output two planning domains $\mathcal{M}$ and $\mathcal{M}^-_?$. The former is such that $\mathcal{M}(\mathcal{C})$ correctly and integrally approximates $\mathcal{M}'(\mathcal{C})$ from the state of the environment when OLAM terminates. The latter correctly approximates $\mathcal{M}'$.

We adopt the following notations. Let $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ and $\boldsymbol{c} = \langle c_1, \ldots, c_n \rangle$ two n-tuple of distinct parameters and constants. If p is an m-ary predicate, $p(\boldsymbol{x})$ denotes an atom $p(x_{i_1}, \ldots, x_{i_m})$ for some m-tuple of indexes $1 \leq i_1, \ldots, i_m \leq n$; and $p(\boldsymbol{c})$ the atom obtained by replacing x_i with c_i in $p(\boldsymbol{x})$. In the following the indexing will be left implicit. OLAM incrementally builds the following sets:

1. $\mathsf{pre}(op)$, which contains the preconditions of the operator op; it is initialized to the whole set of lifted atoms (line 2); an atom $p(\boldsymbol{x})$ is removed from $\mathsf{pre}(op)$ whenever an instance $op(\boldsymbol{c})$ of op is executed successfully in a state s and $p(\boldsymbol{c}) \notin s$ (line 19).
2. $\mathsf{eff}^+_!(op)$ and $\mathsf{eff}^-_!(op)$, which contains the set of lifted positive and negative effects of op learned by the agent; they are initially empty (line 3); a lifted atom $p(\boldsymbol{x})$ is added to $\mathsf{eff}^+_!(op)$ (resp. $\mathsf{eff}^-_!(op)$) if the execution of an instance $op(\boldsymbol{c})$ of op in state s makes $p(\boldsymbol{c})$ become true (resp. false) in the resulting state (lines 20-21).
3. $\mathsf{eff}^+_?(op)$ and $\mathsf{eff}^-_?(op)$, which are sets of lifted atoms that could become part of the positive or negative effects of op; they are initialized to the entire set of lifted atoms (line 2); a lifted atom $p(\boldsymbol{x})$ is removed from $\mathsf{eff}^+_?(op)$ (resp. $\mathsf{eff}^-_?(op)$) if $p(\boldsymbol{x})$ is discovered to be a positive or negative effect or if the atom $p(\boldsymbol{c})$ is false (resp. true) in a state s and remains false (resp. true) after executing successfully an instance $op(\boldsymbol{c})$ of op in s (lines 22-23).
4. $\mathsf{pre}_\perp(op)$, which is a set of sets of lifted preconditions for op such that in every non empty set in $\mathsf{pre}_\perp(op)$ there is at least one precondition of op; $\mathsf{pre}_\perp(op)$ is initialized to a set including only the empty set (line 4); $\mathsf{pre}_\perp(op)$ is augmented by the set formed by any lifted fact $p(\boldsymbol{x})$ such that $p(\boldsymbol{c})$ is false in a state s, if the execution of an instance $op(\boldsymbol{c})$ of op fails in s (line 26).
5. $\mathsf{eff}^+_{!?}(op)$ and $\mathsf{eff}^-_{!?}(op)$, which are derived sets denoting $\mathsf{eff}^+_!(op) \cup \mathsf{eff}^+_?(op)$ and $\mathsf{eff}^-_!(op) \cup \mathsf{eff}^-_?(op)$.

Let denote the sets of preconditions and positive/negative effects of any operator op of $\mathcal{M}'$ by $\mathsf{pre}'(op)$ and $\mathsf{eff}'^{+/-}(op)$, respectively. The update of the sets built by OLAM guarantees that $\mathsf{pre}(op)$ is a superset of $\mathsf{pre}'(op)$, $\mathsf{eff}^{+/-}_!(op)$ are subsets of $\mathsf{eff}'^{+/-}(op)$, and $\mathsf{eff}^{+/-}_{!?}(op)$ are superset of $\mathsf{eff}'^{+/-}_{!?}(op)$.

At each iteration of the external loop (lines 7–31), the agent selects a state s' and a ground action $op'(\boldsymbol{c}')$. s' is reachable from the current state with the model $\mathcal{M}$ learned so far; the ground action $op'(\boldsymbol{c}')$ is such that its execution in s' could provide to the agent some additional information about the preconditions, the positive, or the negative effects of op'. This condition is formalised by (2)–(4). In particular, if condition (2) holds, the preconditions of op' could be refined by executing $op'(\boldsymbol{c}')$ in s', because s' does not contain all the preconditions of $op'(\boldsymbol{c}')$. Indeed if $op'(\boldsymbol{c}')$ will succeed, then the preconditions which are false in s' can be eliminated. If condition (3) (resp. (4)) holds, some positive (resp. negative) effects of op' could be learned, because $op'(\boldsymbol{c}')$ is executable in s' and s' does not contain all the facts in $\mathsf{eff}^+_?(op'(\boldsymbol{c}'))$ (resp. contains at least a fact in $\mathsf{eff}^-_?(op'(\boldsymbol{c}'))$). Indeed, the facts in $\mathsf{eff}^+_?(op'(\boldsymbol{c}'))$ but not in s' which become true can be added to the positive effects. Similarly, the facts that are in

$\mathsf{eff}^-_?(op'(\boldsymbol{c}'))$) and in s' which become false can be added to the negative effects. The selection of such a state s' and action $op'(\boldsymbol{c}')$ is done by constructing a plan from the current state s to a state s' which satisfies conditions (2)–(4) for an $op'(\boldsymbol{c}')$ (line 8). If there is more than one action $op'(\boldsymbol{c}')$ that satisfies conditions (2)–(4) in s', one of them is randomly selected. The choice of s', $op'(c')$ and the associated plan is obtained by invoking PLAN with the following goal:

$$G = \bigvee_{\substack{op(\boldsymbol{c})\in\mathcal{A} \\ P^+P^-E^+E^- \text{ satisfy } (i\text{–}vi)}} \left(\bigwedge_{p(\boldsymbol{c})\in P^+\cup E^-} p(\boldsymbol{c}) \;\wedge \bigwedge_{p(\boldsymbol{c})\in P^-\cup E^+} \neg p(\boldsymbol{c}) \right) \quad (1)$$

$$(i)\ P^-\cup E^+\cup E^-\neq\emptyset, \quad (ii)\ P^+\cap P^-=\emptyset,$$
$$(iii)\ P^+\cup P^- = \mathsf{pre}(op(\boldsymbol{c})), (iv)\ P^-\notin \mathsf{pre}_\perp(op(\boldsymbol{c}))\setminus\{\emptyset\},$$
$$(v)\ E^+\subseteq \mathsf{eff}^+_?(op(\boldsymbol{c})), \quad (vi)\ E^-\subseteq \mathsf{eff}^-_?(op(\boldsymbol{c})).$$

Each disjunct in (1) describes a set of states from which the agent can potentially learn something by executing $op(\boldsymbol{c})$. P^+ and P^- partition the preconditions of $op(\boldsymbol{c})$ so that the atoms in P^+ are true in s', the atoms in P^- are false in s', and set P^- has not already been checked to be necessary for successfully executing $op(\boldsymbol{c})$. E^+ is a subset of possible positive effects of $op(\boldsymbol{c})$ which are false in s' and can become true by executing $op(\boldsymbol{c})$; similarly for E^-. Notice that for every state s' that contains P^+ and E^- and does not contain P^- and E^+, and every action $op'(\boldsymbol{c}')$ such that (iv) and (v) and (vi) hold, when condition (2) is satisfied by s' and $op'(\boldsymbol{c}')$, P^- is not empty; when condition (3) is satisfied by s' and $op'(\boldsymbol{c}')$, E^+ is not empty; finally, when condition (4) is satisfied by s' and $op'(\boldsymbol{c}')$, E^- is not empty.

In the internal loop (lines 9–30), OLAM executes π and if it manages to successfully complete the execution of π (i.e., $\pi=\langle\rangle$, line 10) the ground action $op'(\boldsymbol{c}')$ will be executed in the environment where the agent acts (line 17). The dynamics of such an environment is unknown by the agent, and it determines the result returned by call EXECUTE$(op(\boldsymbol{c}))$. When a ground action $op(\boldsymbol{c})$ is successfully executed, OLAM observes the state of the environment s_{next} resulting from the execution (line 18), and updates sets $\mathsf{pre}(op)$, $\mathsf{eff}^{+/-}_!(op)$ and $\mathsf{eff}^{+/-}_?(op)$ according to the criteria defined above (lines 19–23). If the $op(\boldsymbol{c})$ execution fails in the environment, $\mathsf{pre}_\perp(op)$ is extended as described above, and π is reset to nil since its execution deviates from the expected trajectory computed according to the domain $\mathcal{M}$ learned so far (lines 26-27).

Theorem 1 (Termination)**.** *Algorithm OLAM terminates.*

Proof sketch. The algorithm terminates because at every iteration of the external loop (7–31) one of the following facts holds for some operator op: the size of $\mathsf{pre}(op)$, $\mathsf{eff}^+_?(op)$, or $\mathsf{eff}^-_?(op)$ is reduced; the size of $\mathsf{pre}_\perp(op)$, $\mathsf{eff}^+_!(op)$, or $\mathsf{eff}^-_!(op)$ is increased. Moreover, $\mathsf{pre}_\perp(op)$ cannot be larger than $2^{\mathcal{P}(\mathsf{par}(op))}$, all the other sets cannot be larger than $\mathcal{P}(\mathsf{par}(op))$, and the number of operators is equal to $|\mathcal{O}|$. □

In the rest of the section, we study the properties of correctness and integrity for the learned models $\mathcal{M}$ and $\mathcal{M}^-_?$.

Theorem 2 (Correctness of $\mathcal{M}^-_?$)**.** *$\mathcal{M}^-_?$ correctly approximates $\mathcal{M}'$.*

Algorithm 1 OLAM

Require: $\mathcal{M}=\langle\mathcal{P},\mathcal{O},\{\mathsf{par}(op),\emptyset,\emptyset\}_{op\in\mathcal{O}}\rangle, \mathcal{C}$
1: $s\leftarrow$ OBSERVE()
2: $\forall op\in\mathcal{O}, \mathsf{eff}^-_?(op)\leftarrow\mathsf{eff}^+_?(op)\leftarrow\mathsf{pre}(op)\leftarrow\mathcal{P}(\mathsf{par}(op))$
3: $\forall op\in\mathcal{O}, \mathsf{eff}^-_!(op)\leftarrow\mathsf{eff}^+_!(op)\leftarrow\emptyset$
4: $\forall op\in\mathcal{O}, \mathsf{pre}_\perp(op)\leftarrow\{\emptyset\}$
5: $\mathcal{M}\leftarrow\langle\mathcal{P},\mathcal{O},\{\mathsf{par}(op),\mathsf{pre}(op),\mathsf{eff}^+_!(op),\mathsf{eff}^-_!\}_{op\in\mathcal{O}}\rangle$
6: $\pi\leftarrow nil$
7: **while** $\exists s', op'(\boldsymbol{c}')$ such that s' is reachable from s by $\mathcal{M}(\mathcal{C})$ and (2) $\vee$ (3) $\vee$ (4) holds **do**
8: $\quad\pi\leftarrow$ PLAN$(\mathcal{M}(\mathcal{C}),s,s')$
9: $\quad$**while** $\pi\neq nil$ **do**
10: $\quad\quad$**if** $\pi\neq\langle\rangle$ **then**
11: $\quad\quad\quad op(\boldsymbol{c})\leftarrow$ POP(π)
12: $\quad\quad$**else**
13: $\quad\quad\quad op(\boldsymbol{c})\leftarrow op'(\boldsymbol{c}')$
14: $\quad\quad\quad\pi\leftarrow nil$
15: $\quad\quad$**end if**
16: $\quad\quad\boldsymbol{x}\leftarrow\mathsf{par}(op)$
17: $\quad\quad$**if** EXECUTE$(op(\boldsymbol{c}))$ does not fail **then**
18: $\quad\quad\quad s_{next}\leftarrow$ OBSERVE()
19: $\quad\quad\quad\mathsf{pre}(op)\leftarrow\{p(\boldsymbol{x})\in\mathsf{pre}(op)\mid p(\boldsymbol{c})\in s\}$
20: $\quad\quad\quad\mathsf{eff}^+_!(op)\leftarrow\mathsf{eff}^+_!(op)\cup\{p(\boldsymbol{x})\mid p(\boldsymbol{c})\in s_{next}\setminus s\}$
21: $\quad\quad\quad\mathsf{eff}^-_!(op)\leftarrow\mathsf{eff}^-_!(op)\cup\{p(\boldsymbol{x})\mid p(\boldsymbol{c})\in s\setminus s_{next}\}$
22: $\quad\quad\quad\mathsf{eff}^+_?(op)\leftarrow\mathsf{eff}^+_?(op)\setminus\{p(\boldsymbol{x})\mid p(\boldsymbol{c})\notin s\cap s_{next}\}$
23: $\quad\quad\quad\mathsf{eff}^-_?(op)\leftarrow\mathsf{eff}^-_?(op)\setminus\{p(\boldsymbol{x})\mid p(\boldsymbol{c})\in s\cup s_{next}\}$
24: $\quad\quad\quad s\leftarrow s_{next}$
25: $\quad\quad$**else**
26: $\quad\quad\quad\mathsf{pre}_\perp(op)\leftarrow\mathsf{pre}_\perp(op)\cup\{\{p(\boldsymbol{x})\in\mathsf{pre}(op)\mid p(\boldsymbol{c})\notin s\}\}$
27: $\quad\quad\quad\pi\leftarrow nil$
28: $\quad\quad$**end if**
29: $\quad\quad\mathcal{M}\leftarrow\langle\mathcal{P},\mathcal{O},\{\mathsf{par}(op),\mathsf{pre}(op),\mathsf{eff}^+_!(op),\mathsf{eff}^-_!\}_{op\in\mathcal{O}}\rangle$
30: $\quad$**end while**
31: **end while**
32: $\mathcal{M}^-_?\leftarrow\langle\mathcal{P},\mathcal{O},\{\mathsf{par}(op),\mathsf{pre}(op),\mathsf{eff}^+_!(op),\mathsf{eff}^-_{!?}\}_{op\in\mathcal{O}}\rangle$
33: **return** $\mathcal{M},\mathcal{M}^-_?$

Conditions in line 7:

$$\mathsf{pre}(op'(\boldsymbol{c}'))\setminus s'\notin\mathsf{pre}_\perp(op'(\boldsymbol{c}')) \quad (2)$$
$$\mathsf{pre}(op'(\boldsymbol{c}'))\subseteq s' \text{ and } \mathsf{eff}^+_?(op'(\boldsymbol{c}'))\not\subseteq s' \quad (3)$$
$$\mathsf{pre}(op'(\boldsymbol{c}'))\subseteq s' \text{ and } \mathsf{eff}^-_?(op'(\boldsymbol{c}'))\cap s'\neq\emptyset \quad (4)$$

Proof sketch. The correctness of $\mathcal{M}^-_?$ derives from the fact that at every execution step of OLAM, the set of preconditions of every operator op in $\mathcal{M}^-_?$ is a superset of $\mathsf{pre}'(op)$, the set of positive effects of op is a subset of $\mathsf{eff}'^+(op)$, and finally the set of negative effects of op is a superset of $\mathsf{eff}'^-(op)$. □

Theorem 3 (Correctness of $\mathcal{M}$)**.** *$\mathcal{M}(\mathcal{C})$ correctly approximates $\mathcal{M}'(\mathcal{C})$ from the final state of OLAM.*

Proof sketch. The learned model $\mathcal{M}$ is the same as $\mathcal{M}^-_?$ but the set of negative effects of every operator op of $\mathcal{M}$ is $\mathsf{eff}^-_!(op)$. The statement of the theorem derives from the fact that op cannot have any missing negative effect that is true in some state s reachable from the final state of OLAM. Indeed, by construction, $\mathsf{eff}^-_!(op)\subseteq\mathsf{eff}'^-(op)\subseteq\mathsf{eff}^-_{!?}(op)$, and since condition (4) is false, $\mathsf{eff}^-_?(op(\boldsymbol{c}))\cap s=\emptyset$ is true. □

Theorem 4 (Integrity of $\mathcal{M}$)**.** *$\mathcal{M}(\mathcal{C})$ integrally approximates*

$\mathcal{M}'(\mathcal{C})$ from the final state of OLAM*.*

Proof sketch. The statement of the theorem derives from the fact that (i) every operator op cannot have any missing positive effect that is false in some state s reachable from the final state of OLAM, and (ii) if an instance $op(\boldsymbol{c})$ of op is executable from s via $\mathcal{M}'(\mathcal{C})$ then it is also executable from s via $\mathcal{M}(\mathcal{C})$. Statement (i) derives from the facts that, by construction, $\mathsf{eff}^+_!(op) \subseteq \mathsf{eff}'^+(op) \subseteq \mathsf{eff}^+_{!?}(op)$, and since condition (3) is false, $\mathsf{eff}^+_?(op(\boldsymbol{c})) \subseteq s$ is true. Statement (ii) derives from the fact that, since condition (2) is false, $\mathsf{pre}(op(\boldsymbol{c})) \setminus s \in \mathsf{pre}_\perp(op(\boldsymbol{c}))$, and at any execution step, if $\phi \in \mathsf{pre}_\perp(op)$ and $\phi \neq \emptyset$ then $\phi \cap \mathsf{pre}'(op) \neq \emptyset$. □

The learned model $\mathcal{M}$ approximates the GTM from the final state s_f of OLAM *both* correctly and integrally. This implies that all and only the valid plans computed from s_f via $\mathcal{M}$ are valid plans from s_f via the GTM. Therefore, if a complete algorithm fails to reach a given set of goals from s_f via $\mathcal{M}$, then the goals cannot be reached also via the GTM.

5 Experiments

We evaluate the effectiveness of OLAM for online learning planning domains on 23 planning domains, including the domains from the learning tracks of the past IPCs and the domains used by Aineto *et al.* (2019). For each domain, using an available problem generator, we randomly generated 10 small or middle-size instances with a number of objects ranging from 3 to 241 and consequently a number of potential grounded actions ranging from 12 to about $3.16 \cdot 10^6$. For every domain OLAM is run on all the generated problem instances, from the smallest to the largest. On the first instance, OLAM takes as input the empty set of preconditions, positive and negative effects; for the successive runs, OLAM takes as input the planning domain $\mathcal{M}$ learned at the previous run. In OLAM, the calls EXECUTE and OBSERVE (lines 17-18) are implemented by a simulator of the IPC domain. The transition function of such a model is not known by the agent, who can only ask to execute actions and observe the current state. For function PLAN of Algorithm 1 (line 8), we adopt FAST-DOWNWARD [Helmert, 2006] with a 60 seconds timeout. All experiments were run on an Intel Xeon Skylake 2.3 GHz with 8 cores and 64 GB of RAM.

The learned planning domain is compared with the GTM, as done by Aineto *et al.* (2019), by precision and recall measures. Given a learned model $\mathcal{M}$ and GTM $\mathcal{M}'$, we define precision and recall for preconditions (P_{pre}, R_{pre}), positive and negative effects (P_{eff^-}, P_{eff^+}, R_{eff^-}, R_{eff^+}). Specifically, P_{pre} and R_{pre} are defined as follows:

$$P_{\mathsf{pre}} = \frac{\sum_{op}|\mathsf{pre}(op) \cap \mathsf{pre}'(op)|}{\sum_{op}|\mathsf{pre}(op)|} \quad R_{\mathsf{pre}} = \frac{\sum_{op}|\mathsf{pre}(op) \cap \mathsf{pre}'(op)|}{\sum_{op}|\mathsf{pre}'(op)|}.$$

Intuitively, they measure the (relative) amount of *extra* learned preconditions w.r.t. the GTM, and the (relative) amount of *missing* preconditions w.r.t. the GTM, respectively. The lower these amounts, the greater the measures. Similarly we define precision and recall for eff^- and eff^+. If the precision and recall measures for pre, eff^- and eff^+ is 1, then the learned model is exactly the same as in the GTM for pre, eff^- and eff^+, respectively. The *overall* precision P and recall R are defined considering pre, eff^-, eff^+ together. I.e.,

$$P = \frac{\sum_{op}|\mathsf{pre}(op) \cap \mathsf{pre}(op)'| + |\mathsf{eff}^+(op) \cap \mathsf{eff}'^+(op)'| + |\mathsf{eff}^-(op) \cap \mathsf{eff}'^-(op)'|}{\sum_{op}|\mathsf{pre}(op)| + |\mathsf{eff}^+(op)| + |\mathsf{eff}^-(op)|},$$

and similarly for R.

Domain	$\#I$	P_{pre}	R_{pre}	P_{eff^+}	R_{eff^+}	P_{eff^-}	R_{eff^-}	P	R
barman	4	0.95	1	1	1	1	1	0.97	1
blocksworld	1	1	1	1	1	1	1	1	1
depots	1	0.94	1	1	1	1	1	0.97	1
driverlog	2	0.88	1	1	1	1	1	0.93	1
elevators	3	0.81	1	1	1	1	1	0.88	1
ferry	1	0.88	1	1	1	1	1	0.94	1
floortile	1	0.71	1	1	1	1	1	0.83	1
gold-miner	2	0.68	1	1	1	1	1	0.80	1
grid	2	0.71	1	1	1	1	1	0.82	1
gripper	1	1	1	1	1	1	1	1	1
hanoi	1	0.80	1	1	1	1	1	0.88	1
matching-bw	3	0.97	1	1	1	1	1	0.99	1
miconic	1	1	1	1	1	1	1	1	1
n-puzzle	1	0.75	1	1	1	1	1	0.88	1
nomystery	1	0.75	1	1	1	1	1	0.85	1
parking	2	0.78	1	1	1	1	1	0.89	1
rover	5	0.78	1	1	0.65	1	0.54	0.83	0.84
satellite	1	1	1	1	1	1	1	1	1
sokoban	1	0.80	1	1	1	1	1	0.89	1
spanner	1	0.90	1	1	1	1	1	0.94	1
tpp	3	0.94	1	1	1	1	1	0.97	1
transport	1	0.91	1	1	1	1	1	0.95	1
zenotravel	1	1	1	1	1	1	1	1	1

Table 1: Number of instances used to learn $\mathcal{M}$ (column 2), precision and recall over the preconditions, positive and negative effects of $\mathcal{M}$ (columns 3–8), overall precision and recall of $\mathcal{M}$ (columns 9-10).

Table 1 summarizes the efficacy of $\mathcal{M}$ w.r.t. the GTM in terms of precision and recall. By construction of sets $\mathsf{pre}(op)$ and $\mathsf{eff}^{+/-}_!(op)$ of every operator op, R_{pre}, P_{eff^+}, and P_{eff^-} of $\mathcal{M}$ must be equal to 1, i.e., there is no missing precondition and extra effect in the learned model $\mathcal{M}$ w.r.t. the GTM. This is confirmed by the results in Table 1. Moreover, P_{pre} is always quite high, although usually lower than 1, i.e., there are few extra preconditions in the learned model w.r.t. the GTM. The extra learned preconditions are static predicates such that, when the action is grounded, the corresponding grounded preconditions are true in all the states reachable from the initial state. This prevents the remotion of these extra preconditions from a correct learned model, like $\mathcal{M}$. The recall over the positive/negative effects is always equal to 1 for every domain but ROVER, i.e., there are no missing effects (except for ROVER) in the learned model w.r.t. the GTM.

The results in Table 1 also show that domain $\mathcal{M}$ can be learned using very few problems, often using only a single problem. Note that such a domain is learned by few small problems, and it does not mention their constants, i.e., it is general and hence suitable even for much larger problems. This shows that overall OLAM is able to effectively generalize between the experience derived from small environments and the future experience in large environments.

We also study the efficacy of $\mathcal{M}^-_?$ w.r.t. the GTM. The difference between the learned models $\mathcal{M}$ and $\mathcal{M}^-_?$ consists in the fact that $\mathcal{M}^-_?$ also includes set $\mathsf{eff}^-_?(op)$ as negative effects of an operator op. Therefore, the precision and the recall over the preconditions and the positive effects of $\mathcal{M}^-_?$ are

	with assumption				without assumption			
Domain	P_{eff^-}	R_{eff^-}	P	R	P_{eff^-}	R_{eff^-}	P	R
barman	1	1	0.97	1	0.24	1	0.56	1
blocksworld	1	1	1	1	0.43	1	0.69	1
depots	1	1	0.97	1	0.56	1	0.80	1
driverlog	1	1	0.93	1	0.23	1	0.53	1
elevators	1	1	0.88	1	0.15	1	0.42	1
ferry	1	1	0.94	1	0.50	1	0.75	1
floortile	1	1	0.83	1	0.10	1	0.29	1
gold-miner	1	0.82	0.80	0.95	0.18	1	0.41	1
grid	1	1	0.82	1	0.28	1	0.55	1
gripper	1	1	1	1	1	1	1	1
hanoi	1	1	0.88	1	1	1	0.88	1
matching-bw	1	1	0.99	1	0.32	1	0.65	1
miconic	1	1	1	1	0.23	1	0.62	1
n-puzzle	1	1	0.88	1	0.50	1	0.70	1
nomystery	1	1	0.85	1	0.10	1	0.30	1
parking	1	1	0.89	1	0.35	1	0.60	1
rover	1	0.54	0.83	0.84	0.16	0.54	0.55	0.84
satellite	1	1	1	1	0.67	1	0.92	1
sokoban	1	1	0.89	1	0.25	1	0.53	1
spanner	1	1	0.94	1	0.40	1	0.70	1
tpp	1	1	0.97	1	0.15	1	0.42	1
transport	1	1	0.95	1	0.33	1	0.65	1
zenotravel	1	1	1	1	0.33	1	0.67	1

Table 2: Precision and recall over the negative effects of $\mathcal{M}^-_?$ and overall model $\mathcal{M}^-_?$ with the assumption $\mathsf{eff}'^-(op) \subseteq \mathsf{pre}'(op)$ (columns 2–5), and without this assumption (columns 6–9).

the same as in Table 1. Table 2 gives the precision and recall over the negative effects of $\mathcal{M}^-_?$ and over all domain $\mathcal{M}^-_?$. For this study we consider $\mathcal{M}^-_?$ with and without assuming $\mathsf{eff}'^-(op) \subseteq \mathsf{pre}'(op)$, i.e., when this assumption is made, the atoms in $\mathsf{eff}^-_?(op)$ that are not in the preconditions of an operator are removed. By construction of set $\mathsf{eff}^-_{!?}$, R_{eff^-} must be equal to 1. Surprisingly, this is false for domains GOLD-MINER and ROVER. The reason why this happens is that for these domains an assumption of ours does not hold: ROVER is a special domain including operators with inconsistent effects, i.e., $\mathsf{eff}'^+(op) \cap \mathsf{eff}'^-(op) \neq \emptyset$, for some operators. For GOLD-MINER, the assumption $\mathsf{eff}'^-(op) \subseteq \mathsf{pre}'(op)$ does not hold. This assumption is violated also in domains PARKING, SATELLITE and MATCHING-BW, but for these domains there is no missing negative effect in $\mathcal{M}^-_?$, since OLAM on line 21 learns $\mathsf{eff}^-_!$ regardless of this assumption. Interestingly, P^-_{eff} with this assumption is always equal to 1, while without the assumption it is almost always quite low. This gap gives evidence that such an assumption can be very useful for removing extra negative effects from the learned domain.

We compare OLAM with a version of the algorithm that explores the world randomly. The random strategy reaches an average precision and recall of 0.45 and 0.63, against the average precision and recall of 0.99 and 0.92 obtained by OLAM, which shows that the generation of informative plan traces is extremely helpful.

In the last experiment we compare the online learning of OLAM with the offline learning method proposed by Fama [Aineto *et al.*, 2019]. Fama takes as input a set of plans with their state trajectories. Since OLAM does not support partial observability, we set Fama for working in a fully observable environment, and considered the same sets of plan traces and planning domains (but VISITALL and ZENOTRAVEL) as in [Aineto *et al.*, 2019]. The set of plan traces consists of 10 traces with 10 states; the set of planning domains does not contain ZENOTRAVEL and VISITALL, because the distributed version of Fama finds no solution for ZENOTRAVEL, and there is no problem generator available for VISITALL. Since Fama adopts the assumption $\mathsf{eff}'^-(op) \subseteq \mathsf{pre}'(op)$ for any operator op, we compared the planning domain derived from Fama with $\mathcal{M}^-_?$ using the same assumption. We obtained similar results from the comparison between Fama and the other learned domain $\mathcal{M}$.

	OLAM			Fama			
Domain	Time	P	R	Time	P	R	Δ acts
blocksworld	5.03	**1**	**1**	510	**1**	**1**	-80
driverlog	20.42	**0.93**	**1**	349	0.79	0.85	-43
ferry	7.54	**0.94**	**1**	267	0.80	0.93	-85
floortile	47.34	**0.83**	**1**	517	0.82	0.78	-15
grid	36.92	**0.82**	**1**	306	0.81	0.74	-1
gripper	3.50	**1**	**1**	165	0.86	0.93	-89
hanoi	2.38	**0.88**	**1**	818	**0.88**	0.86	-96
miconic	4.24	**1**	**1**	200	0.81	**1**	-78
n-puzzle	1.97	**0.88**	**1**	23	0.86	**1**	-91
parking	183.94	**0.89**	**1**	895	0.84	0.84	-47
rover	154.10	**0.83**	**0.84**	629	0.51	0.53	175
satellite	11.26	**1**	**1**	65	0.70	0.89	-54
transport	74.98	**0.95**	**1**	280	0.80	0.89	-32

Table 3: CPU-seconds, precision and recall of OLAM (columns 2–4) and Fama (columns 5–7); difference between number of actions executed by Fama and OLAM (column 8): negative values mean that OLAM executes fewer actions. Bold values indicate best results.

Table 3 compares OLAM and Fama. OLAM obtains better or equal precision and recall, and generally it is also much faster. In all the domains but ROVER, OLAM executes less actions than Fama. We think that the difference for ROVER is related to the consistent-effects assumption made in OLAM that in ROVER does not hold. Overall, learning the planning domain online is much more effective than learning it offline. In our online approach, indeed, the agent selects the goals to reach and actions to execute to optimize learning, while in offline approaches actions are provided in the input traces.

6 Conclusions

The paper proposes an online algorithm, OLAM, for learning PDDL planning domain under the assumption of full observability. OLAM incrementally learns an action model by selecting goals to reach and actions to execute that allow to acquire useful information about the operators. The paper shows some important theoretical properties of OLAM concerning the completeness and integrity of the learned models. An implementation of OLAM shows good learning performance on a large set of benchmarks from the IPCs, and outperforms a state-of-the-art method for learning action models offline. OLAM works with full observability; extension to partial observability is part of the future work.

Acknowledgements

This work is partially supported by the EU ICT-48 2020 project TAILOR (No. 952215).

References

[Aineto *et al.*, 2018] D. Aineto, S. Jiménez, and E. Onaindia. Learning strips action models with classical planning. In *ICAPS*, 2018.

[Aineto *et al.*, 2019] Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artif. Intell.*, 275:104 – 137, 2019.

[Amir and Chang, 2008] Eyal Amir and Allen Chang. Learning partially observable deterministic action models. *J. Artif. Intell. Res.*, 33:349–402, 2008.

[Bonet and Geffner, 2020] Blai Bonet and Hector Geffner. Learning first-order symbolic representations for planning from the structure of the state space. In *ECAI*, 2020.

[Certicky, 2014] Michal Certicky. Real-time action model learning with online algorithm 3SG. *Applied Artificial Intelligence*, 28(7):690–711, 2014.

[Cresswell *et al.*, 2013] Stephen Cresswell, Thomas Leo McCluskey, and Margaret Mary West. Acquiring planning domain models using *LOCM*. *Knowledge Eng. Review*, 28(2):195–213, 2013.

[Fern *et al.*, 2004] Alan Fern, Sung Wook Yoon, and Robert Givan. Learning domain-specific control knowledge from random walks. In *ICAPS*, 2004.

[García-Martínez and Borrajo, 2000] Ramón García-Martínez and Daniel Borrajo. An integrated approach of learning, planning, and execution. *J. Intell. Robotic Syst.*, 29(1):47–78, 2000.

[Gil, 1994a] Yolanda Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *ICML*, 1994.

[Gil, 1994b] Yolanda Gil. Learning new planning operators by exploration and experimentation. In *AAAI*, 1994.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

[Kitano and Tadokoro, 2001] Hiroaki Kitano and Satoshi Tadokoro. Robocup rescue: A grand challenge for multiagent and intelligent systems. *AI Mag.*, 22(1):39–52, 2001.

[Lamanna *et al.*, 2021] Leonardo Lamanna, Alfonso Gerevini, Alessandro Saetti, Luciano Serafini, and Paolo Traverso. On-line learning of planning domains from sensor data in pal: Scaling up to large state spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11862–11869, 2021.

[Mourão *et al.*, 2012] Kira Mourão, Luke S. Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *UAI*, 2012.

[Newton *et al.*, 2007] Muhammad Abdul Hakim Newton, John Levine, Maria Fox, and Derek Long. Learning macro-actions for arbitrary planners and domains. In *ICAPS*, 2007.

[Rodrigues *et al.*, 2010a] Christophe Rodrigues, Pierre Gérard, and Céline Rouveirol. Incremental learning of relational action models in noisy environments. In *ILP*, 2010.

[Rodrigues *et al.*, 2010b] Christophe Rodrigues, Pierre Gérard, Céline Rouveirol, and Henry Soldano. Incremental learning of relational action rules. In *ICMLA*, 2010.

[Rodrigues *et al.*, 2011] Christophe Rodrigues, Pierre Gérard, Céline Rouveirol, and Henry Soldano. Active learning of relational action models. In *ILP*, 2011.

[Stachniss *et al.*, 2016] Cyrill Stachniss, John J. Leonard, and Sebastian Thrun. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, Springer Handbooks, pages 1153–1176. Springer, 2016.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[Walsh and Littman, 2008] Thomas J Walsh and Michael L Littman. Efficient learning of action schemas and web-service descriptions. In *AAAI*, 2008.

[Wang, 1996] Xuemei Wang. Planning while learning operators. In *AAAI*, 1996.

[Xu and Laird, 2010] Joseph Z. Xu and John E. Laird. Instance-based online learning of deterministic relational action models. In *AAAI*, 2010.

[Xu and Laird, 2011] Joseph Z. Xu and John E. Laird. Combining learned discrete and continuous action models. In *AAAI*, 2011.

[Xu and Laird, 2013] Joseph Zhen Ying Xu and John E. Laird. Learning integrated symbolic and continuous action models for continuous domains. In *AAAI*, 2013.

[Yang *et al.*, 2007] Q Yang, K Wu, and Y Jang. Learning action models from plan examples using weighted max-sat. *Artif. Intell.*, 171:107–143, 2007.

[Zhuo and Kambhampati, 2013] Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *IJCAI*, 2013.

[Zhuo *et al.*, 2010] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. Learning complex action models with quantifiers and logical implications. *Artif. Intell.*, 174(18):1540–1569, 2010.