On-line Learning of Planning Domains from Sensor Data in PAL: Scaling up to Large State Spaces

Leonardo Lamanna,[1,2] **Alfonso E. Gerevini,**[1] **Alessandro Saetti,**[1] **Luciano Serafini,**[2] **Paolo Traverso**[2]
[1]Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Italy
[2]Fondazione Bruno Kessler, Trento, Italy

Abstract

We propose an approach to learn an extensional representation of a discrete deterministic planning domain from observations in a continuous space navigated by the agent actions. This is achieved through the use of a perception function providing the likelihood of a real-value observation being in a given state of the planning domain after executing an action. The agent learns an extensional representation of the domain (the set of states, the transitions from states to states caused by actions) and the perception function on-line, while it acts for accomplishing its task. In order to provide a practical approach that can scale up to large state spaces, a "sketched" intensional (PDDL-based) model of the planning domain is used to guide the exploration of the environment and learn the states and state transitions. The proposed approach uses a novel algorithm to (i) construct the extensional representation of the domain by interleaving symbolic planning in the PDDL intensional representation and search in the state-transition graph of the extensional representation; (ii) incrementally refine the intensional representation taking into account information about the actions that the agent cannot execute. An experimental analysis shows that the novel approach can scale up to large state spaces, thus overcoming the limits in scalability of the previous work.

1 Introduction

Symbolic planning techniques are based on abstract and most often discrete representations of the world, where the agents perform their actions, usually called *planning domains*. A discrete planning domain is a finite state transition system, i.e., a finite set of states, a finite set of actions, and a transition relation representing how actions lead from states to new states (Ghallab, Nau, and Traverso 2004; 2016). A planning domain can be specified either "extensionally", by explicitly describing each state and each transition (e.g., by a transition matrix), or "intensionally" by means of a planning language, e.g., STRIPS (Fikes and Nilsson 1971) and PDDL (McDermott et al. 1998), whose semantics is given in terms of transition systems. In both cases, the specification of planning domains is a challenging task. A "good" planning domain should be simple enough to make automatic planning feasible, as well as detailed enough to generate effective solution plans (i.e., solutions that can discriminate the states in which the agent has to take different decisions in order to reach a goal). A good planning domain should also abstract away the details of the world state which are irrelevant for the achievement of the agents' goals, keeping only the relevant details. This is useful, e.g., for alleviating the knowledge engineering effort and having planning algorithms perform better.

In many real applications, it may happen that agents do not have a good planning domain in advance. For instance, a robot moving packages among rooms of a building could reason using a planning domain that models the map of the building with a number of flaws or relevant missing details. In these cases, agents should be able to learn and update their models (planning domains) while acting in the world, observing the consequences of their actions, abstracting away the real-world states into some abstract planning state, and updating the state transition relations accordingly. This is the main purpose of the PAL algorithm (Acting and Learning Planning) proposed by Serafini and Traverso (2019a), which learns, incrementally and on-line, a discrete deterministic planning domain from real-value observations of the world. Each state in the domain is linked to observations by the so called *perception function*, which provides the likelihood of the observations when the agent is at that specific state. At each iteration, PAL updates the set of states of the extensional representation of a planning domain, possibly by introducing new states for unexpected observations, and it adjusts the transition relation and the perception function.

A significant drawback of this version of PAL is its poor scalability. Serafini and Traverso (2019a) show that the scalability limits of PAL are due to the fact that planning domains are represented extensionally by explicitly enumerating the set of states that are explored by the agent, without any specific heuristic guidance or reasoning. In order to overcome this limitation of PAL, in this paper we extend it so that to the agent is provided a "drafted" planning domain specified by the well-known language PDDL (McDermott et al. 1998). The domain is drafted in the sense that we require its specification to be initially neither sufficiently detailed nor fully correct, since its main purpose is to *guide the agent in the discovery* of the world and thus also its learning. Furthermore, we provide an algorithm that incrementally updates the initial drafted PDDL domain with additional information about the actions that the agent cannot execute.

In the new system, the presence of both the extensional and the intensional models of the planning domain is exploited to efficiently achieve the agent's goals through alternative planning engines: (i) a shortest-path algorithm for searching a plan in the learned extensional model, and (ii) a PDDL planner for searching a plan in areas of the search space that have not been explored (learned through experience) by the agent yet. An experimental analysis shows that:

- the new resulting method scales up to planning domains that have a large number of states;
- the online learning can significantly reduce the possible flaws in the given intensional planning domain, which asymptotically converges to a correct model;
- exploiting the combined extensional and intensional models of the planning domain through different planners allows the agent to efficiently accomplish its tasks.

In the reminder of the paper, first we formalize the investigated problem, then we propose the new method to address it, provide an experimental evaluation of it, discuss related work, and finally give conclusions.

2 The Plan-Act-Learn Problem

We formalize the Plan-Act-Learn (PAL) problem that was first studied by Serafini and Traverso (2019b) without giving a formal definition. A solution to a PAL-problem instance consists in a learned abstract model of the agents' environment that can be exploited to achieve a set of goals. In the PAL problem, agents perceive the environment through a series of *perceptions*, where a perception is a vector $\mathbf{x} = \langle x_1, x_2, ..., x_n \rangle$ of continuous variables over real numbers, called *perception variables*. We define the environment where agents operate as a non-deterministic infinite-state transition system, called *perceptible environment*.

Definition 1 (Perceptible environment) *A perceptible environment $\mathcal{E}$ is a tuple (Q, A, τ), where $Q \subseteq \mathbb{R}^n$ is a (possibly infinite) set of perceptions, A is a finite set of actions, and $\tau : Q \times A \rightarrow 2^Q$ is a non-deterministic transition function.*

Function τ returns the set of possible perceptions after the execution of an action $a \in A$ in a state $q \in Q$ (and before executing other successive actions). We adopt the notation $\tau(a, X) = \bigcup_{\mathbf{x} \in X} \tau(a, \mathbf{x})$ for $X \subseteq Q$.

Specifying the components of a perceptible environment is typically extremely complicated, and it cannot be done by hand. In the field of planning, a common assumption is that agents act at an *abstract* level. For instance, the behavior of a robot moving packages among rooms of a building can be conveniently determined by a planning domain where each state corresponds to the fact that the robot and packages are at a certain room, and each transition correspond to an abstract action, such as moving the robot among rooms, picking up packages, and putting down them. We define the search space for planning as a deterministic finite-state transition system.

Definition 2 (Extensional model) *An* extensional model *$\mathcal{M}$ of an environment $\mathcal{E} = (Q, A, \tau)$ is a tuple (S, A, γ) where S is a finite set of (abstract) states, and $\gamma : S \times A \rightarrow S$ is a deterministic transition function.*

Given a state $s \in S$ and an action $a \in A$, the function γ outputs the resulting state reached after the execution of a in s. The action space A of the extensional model is the same as of the perceptible environment, which consists of the set of actions agents can perform.

Definition 3 (Perception function) *Given an extensional model $\mathcal{M} = (S, A, \gamma)$ of an environment (Q, A, τ), a perception function ρ for $\mathcal{M}$ is a function $\rho : Q \times S \rightarrow \mathbb{R}^+$ such that for every $s \in S$, $\rho(\mathbf{x}, s) = p(\mathbf{x} \mid s)$, where $p(\mathbf{x} \mid s)$ is a probability density function on Q.*

The extensional model $\mathcal{M}$ and the perception function ρ shares the same set of states S. Given a perception function ρ and a perception $\mathbf{x} \in Q$, we define the function $\rho^* : Q \rightarrow S$ as $\rho^*(\mathbf{x}) = \operatorname{argmax}_{s \in S} \rho(\mathbf{x}, s)$, and similarly $\rho^*(X) = \{\rho^*(\mathbf{x}) \mid \mathbf{x} \in X\}$. Intuitively, ρ^* is the function that discretizes the infinite set of states Q into the finite set of states S.

Definition 4 (Plan) *A* plan *in an extensional model $\mathcal{M} = (S, A, \gamma)$ from state $s \in S$ to state $s' \in S$ is a sequence $(a_1, \ldots, a_m)$ of m actions in A such that $s' = \gamma(a_m, \gamma(a_{m-1}, \ldots, \gamma(a_1, s)))$.*

A perception goal is a perception $\mathbf{x} \in Q$ that, when perceived by the agent, makes it consider the assigned task accomplished.

Definition 5 (PAL-problem) *Given an environment $\mathcal{E}$, the* PAL-problem *consists in learning an extensional model $\mathcal{M}$ and a perception function ρ from $\mathcal{E}$, such that, for every perception $\mathbf{x}_0 \in Q$ and (non-empty) perception goal set $X_g \subseteq Q$, $\mathcal{M}$ has a plan $(a_1, \ldots, a_m)$ from $\rho^\star(\mathbf{x}_0)$ to some state in $\rho^\star(X_g)$ and $\tau(a_m, \tau(a_{m-1}, \ldots, \tau(a_1, \mathbf{x}_0))) \cap X_g \neq \emptyset$.*

It is important to note that the agent does not know the environment $\mathcal{E}$. The only knowledge about the environment that it has is the one observed through the perception variables when executing actions, as the agent can only perceive the environment and observe the action effects after their execution.

3 Solving the PAL Problem

In this section, we introduce an approach to solving the PAL problem that interleaves planning, acting, and learning using a limited amount of prior knowledge for the agent. Our approach is named as the problem it solves, Plan-Act-Learn (PAL). To learn the extensional model, the agent can apply different strategies: a random exploration strategy is not feasible, since, as shown in previous work (Serafini and Traverso 2019b; 2019a), it does not scale to large state spaces. Alternatively, the agent can use some prior *belief* about the environment to decide a plan that will lead to its current goal. Following this idea, we suppose that such a belief is expressed through an *exploration planning domain* $\mathcal{D}^e$ that is specified by a planning language such as PDDL (McDermott et al. 1998). Intuitively, the agent will decide the next action to perform by computing a plan that reaches a state among those in an input set of goal states G^e from

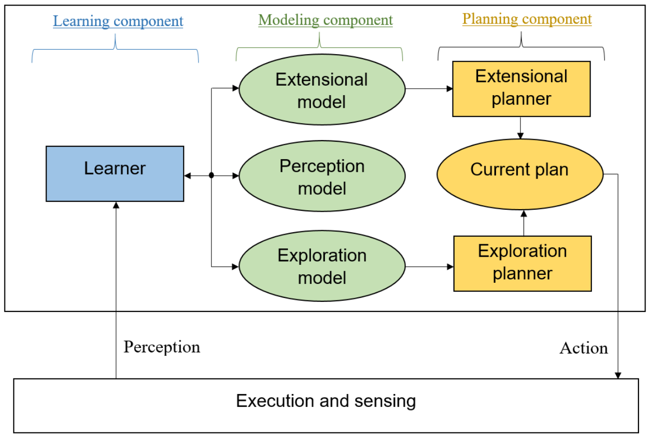

Figure 1: PAL architecture. The square boxes represent modules; the circle ones represent data.

the PDDL state s^e representing the belief of the agent about the current status of the environment (the e index indicates that these are the initial state and set of goal states of the exploration model). Note that we make no assumption about the correctness of $\mathcal{D}^e$; we only need that the transformation used to derive s^e from the current status of the environment is such that the inverse transformation applied to a goal state among those in G^e derives a status of the environment corresponding to a perception in X_g.

The architecture of the proposed solution is shown in Figure 1. The top box of the picture shows the architecture of the PAL agent. It consists of three components: (i) the learner module, (ii) the modeling component, formed by three models, and (iii) the planning component using two kinds of planners. The *learner* updates the perception model using the perceptions from the environment, incrementally constructs the extensional model and revises the exploration model. We call the perception function, together with the history of sensed perceptions, the *perception model*. The exploration model is refined when a failure occurs in the attempt of executing an action. The extensional model and the exploration one are respectively taken as input by the extensional and exploration planner. Firstly, a plan is searched in the extensional model. The search of such a plan may fail because the transition function γ known by the agent at planning time could be incomplete. If no plan is found, the goal-driven exploration strategy is applied by means of an exploration planner. The PAL agent interacts with a simulator (bottom box in the picture), the purpose of which is to simulate the perceptible environment $\mathcal{E}$ where the agent operates, i.e., it simulates the execution of a given action and the sensing of the environment immediately after the action execution. We assume that the simulator knows the comprehensive definition of the transition function τ of $\mathcal{E}$.

The pseudocode of PAL is shown in Algorithm 1. The algorithm takes as input: a perception goal set X_g, a threshold $t \in \mathbb{R}$, an initial perception function ρ_{init}, an initial extensional model $\mathcal{M}_{init}$, and an initial exploration model. The input initial extensional model $\mathcal{M}_{init}$ is composed by a

```
Procedure PAL($X_g, t, \rho_{init}, \mathcal{M}_{init}, \mathcal{D}^e_{init}, s^e, G^e$)
    $\rho, S, \gamma, \mathcal{D}^e, \pi \leftarrow \rho_{init}, S_{init}, \gamma_{init}, \mathcal{D}^e_{init}, \langle\rangle$;
    $\mathbf{x} \leftarrow$ SENSE();
    $S' \leftarrow \{s \in S \mid \rho(\mathbf{x}, s) \geq t\}$;
    if $S' = \emptyset$ then
        $s \leftarrow$ CREATESTATE($\mathbf{x}$);
        $S \leftarrow S \cup \{s\}$;
    end
    else
        $s \leftarrow \rho^\star(\mathbf{x})$;
    end
    $P \leftarrow \langle \mathbf{x}, s\rangle$;   /* The initial history of perceptions*/
    while the CPU time limit is not exceeded do
        if $\mathbf{x} \in X_g$ then
            return Success;   /* $s$ is a goal state */
        end
        if $\pi = \langle\rangle$ then
            $S_g \leftarrow \{s \in S \mid \rho(\mathbf{x}_g, s) \geq t$ and $\mathbf{x}_g \in X_g\}$;
            if $S_g \neq \emptyset$ then
                $\pi \leftarrow$ EXTENSIONALPLANNER($\gamma, s, S_g$);
            end
            if $\pi = \langle\rangle$ then
            /* EXTENSIONALPLANNER has failed */
                $\pi \leftarrow$EXPLORATIONPLANNER($\mathcal{D}^e, s^e, G^e$);
            end
        end
        if $\pi \neq \langle\rangle$ then
            $a \leftarrow$HEAD($\pi$);
            $\pi \leftarrow$TAIL($\pi$);
        end
        else
            Select an action $a \in A$ randomly;
        end
        EXECUTE($a$);
        $\mathbf{x} \leftarrow$SENSE();
        $S' \leftarrow \{s \in S \mid \rho(\mathbf{x}, s) \geq t\}$;
        if $S' = \emptyset$ then
            $s' \leftarrow$ CREATESTATE($\mathbf{x}$);
            $S \leftarrow S \cup \{s'\}$;
            $\gamma \leftarrow \gamma \cup \{(s, a, s')\}$;
            $s^e \leftarrow$UPDATEPDDLSTATE($\mathcal{D}^e, s^e, a$);
        end
        else
            $s' \leftarrow \rho^\star(\mathbf{x})$;
            if $s' = s$ then /* Execution failure */
                $\mathcal{D} \leftarrow$UPDATEPDDLDOMAIN($\mathcal{D}^e, s^e, a$);
                $\pi \leftarrow \langle\rangle$;
            end
            else
                $s^e \leftarrow$UPDATEPDDLSTATE($\mathcal{D}^e, s^e, a$);
            end
        end
        $s \leftarrow s'$;
        $P \leftarrow$APPEND($P, \langle \mathbf{x}, s\rangle$);   /* Update perception history */
        $\rho \leftarrow$UPDATEPERCEPTIONFUNC($\rho, P$);
    end
    return Failure;
```

Algorithm 1: PAL algorithm.

(possibly empty) set of states S_{init}, a (possibly empty) transition function γ_{init}, and a set of actions A that agents can perform; the input initial exploration model is composed by $\mathcal{D}^e_{init}$, s^e, and G^e.

Initially, the agent perceives the environment by sensing perception $\mathbf{x}$ (step 3). Afterwards, it verifies whether there exists at least a state among those in S_{init} such that the likelihood of sensing $\mathbf{x}$ being in this state is greater than a threshold. If it does exist the current state s is set to $\rho^*(\mathbf{x})$, otherwise a new state is created (steps 5–11). The perception history is initialized with the perception $\mathbf{x}$ and the current state s (step 12). If the perception $\mathbf{x}$ belongs to the set X_g, then s is a goal state and the algorithm returns success (steps 14–16). Otherwise, if the plan π is empty, the set of goal states S_g is defined as the states in S which correspond to a goal perception $\mathbf{x}_g \in X_g$ (steps 17–18). A state s corresponds to a goal perception $\mathbf{x}_g$ if $\rho(\mathbf{x}_g, s) \geq t$.

If the set S_g is not empty, the extensional planner is exploited to search a plan π from s to a state in S_g (steps 19–21). Basically, the extensional planner runs the Dijkstra's shortest-path algorithm from the source node representing s in the graph induced by the state transition function γ. If the distance from the source node to a node representing a state among those in S_g is infinity, then there exists no path from the source node to a goal node in the induced graph, and the extensional planner fails. Otherwise, it returns the shortest path from the source node to any goal node. If the extensional planner does not find a plan, the agent searches a plan using the exploration model, i.e., it runs a PDDL planner to achieve goals G^e from s^e, using the PDDL domain $\mathcal{D}^e$ (steps 22–25). Note that, since the exploration model is an approximation of the agent behavior in the real world, it could happen that also this plan does not exist or, if it does exist, an action in the plan is not executable in the real world. If the plan does not exist, an action in A is randomly selected (step 32). Otherwise, the first action of the plan is selected (steps 28-29) and executed (step 34).

After the execution of the action (and before executing the next one), the agent perceives the current environment state (step 35). It is worth noting that the sensing of the same world state can generate different perceptions. This can happen for several reasons. Consider the example of a robot moving packages among rooms of a building, and assume that in this example the GPS coordinates are part of the sensed perception. It may be that the robot is at the lobby of the building more than once, and every time it could be at a different specific position in the lobby. Moreover, even if every time the robot is at the same specific position, the GPS coordinates could be different because of measurement errors of the GPS tracking unit.

Given the last executed action a and the last perception $\mathbf{x}$, the learner updates the perception model, the extensional model and the exploration model. Specifically, if the probability of observing $\mathbf{x}$ from each state in S is lower than threshold t, then the agent creates a new state s', adds s' to S, adds transition $\langle s, a, s'\rangle$ to γ and updates the PDDL state s^e (steps 37–42). The state s^e is updated according to $\mathcal{D}^e$, i.e., adding the positive effects and deleting the negative effects of action a. Otherwise, the agent selects the state s' that maximizes the likelihood of observing $\mathbf{x}$ as the next state (step 44). If the states in S that maximize the likelihood of observing $\mathbf{x}$ are more than one, one of them is randomly selected. If $s = s'$, i.e., the execution of action a fails, then the agent makes plan π empty and updates the PDDL domain $\mathcal{D}^e$ in such a way that action a cannot be executed in state s^e (steps 45–48). If the action has been successfully executed, the PDDL state is updated by applying the action effects (step 50). Finally, the current state s is set to the next state s', the pair $(\mathbf{x}, s)$ is added to the perception history P, and the perception function ρ is updated according to P (steps 53–55). The loop 13–56 is repeated until the CPU-time limit is exceeded. If the loop terminates without having reached a goal state, the algorithm returns failure (step 57).

4 Learning the Perception Function

The perception function ρ allows the agent to map a value $\mathbf{x} = (x_1, \ldots, x_n)$ of n perception variables to the state s^* according to the maximum likelihood criteria $s^* = \mathrm{argmax}_{s_i \in S}\, \rho(s_i, \mathbf{x})$. When the number of perception variables and number of states in the extensional model is high, modelling $\rho(s_i, \mathbf{x})$ with an n-dimensional distribution $p(\mathbf{x} \mid s)$, as proposed by Serafini and Traverso (2019a), is extremely expensive from the computational point of view and result infeasible. A practical simplifying hypothesis can be obtained by assuming that ρ factorizes in n perception functions, one for each perception variable. This means that

$$\rho(s_i, \mathbf{x}) = \prod_{j=1}^{n} \rho_j(s_i, x_j)$$

where each $\rho_j(s_i, x)$ is an unidimensional probability density function. The additional advantage of this factorization is that it allows to associate different thresholds to each perception variable for the same state, instead of a single threshold. We therefore replace the single threshold t in Algorithm PAL with a vector $\boldsymbol{t} = (t_1, \ldots, t_n)$ where $t_i \in \mathbb{R}^+$ is the threshold associated to the i-th perception variable. The set of abstract states on which we have to maximize the likelihood in order to find the next state is thereby defined as

$$S' = \{s_i \in S \mid \rho(s_i, \mathbf{x}) \geq \boldsymbol{t}\}$$

where condition $\rho(s_i, \mathbf{x}) \geq \boldsymbol{t}$ stands for $\bigwedge_{j=1}^{n} \rho_j(s_i, x_j) \geq t_j$ (steps 4, 18, 36 of Algorithm PAL). A concrete, but still very general, model for a single variable perception function which we decided to adopt is the normal distribution. We therefore suppose that, for every state s_i and perception variable j, $\rho_j(s_i, x_j)$ is the normal distribution $\mathcal{N}(x_j \mid \mu_{ij}, \sigma_{ij})$ with mean μ_{ij} and variance σ_{ij}.

The parameters μ_{ij} and σ_{ij} can be learned online (step 55 of Algorithm PAL). Given a sequence of m observations $(\mathbf{x}^{(k)})_{k=1}^{m}$ associated to the same state s_i, the mean μ_{ij} of the j-th perception variable is updated as follows:

$$\mu_{ij} = \frac{2}{m(m+1)} \sum_{k=0}^{m-1} (m-k) x_j^{(m-k)}$$

For each perception variable, its mean in the state s_i is set to the normalized weighted sum of all perception observations associated to the state s_i. The first observation $\mathbf{x}^{(1)}$ is the one associated to the state when it is created; $\mathbf{x}^{(m)}$ is the last perception associated to the state by procedure PAL. The oldest the observation, the less weight is given. We assume that the standard deviation σ_{ij} is known a priori and keeps unchanged since given by the sensors, although in principle our approach could learn it from the data.

The choice of the sequence $\boldsymbol{t}$ is important since it strongly affects the agent capability to correctly build the extensional model. The higher the thresholds the more states are created. With a very low threshold redundant states can be introduced, i.e., states that corresponds to very similar perceptions; from these states agents have to take the same decision to reach a goal state, and hence they should be clustered in the same state. On the other hand, if the thresholds are too low, then more than one abstract state is collapsed in a unique extensional state. A reasonable setting for t_i can be obtained by defining $t_i = \mathcal{N}(2\sigma_{noise,i}|0, \sigma_{noise,i})$, where $\sigma_{noise,i}$ is the maximum measurement noise of the sensor associated to the i-th perception variable.

5 Experimental Analysis

In our experimental analysis we evaluate the effectiveness of the proposed approach and, in particular, the usefulness of using the exploration planner for guiding the search. As exploration planner we used the well-known planner FastDownward (Helmert 2006). All experimental tests were conducted on an Intel Xeon Skylake 2.3 GHz with 8 cores and 128 GB of RAM. The time limit for each run of PAL was 60 minutes, after which termination was forced.

Benchmarks and simulators Our benchmarks derive from three well-known planning domains: Logistics, Grid, and Rovers. We assume that these domains are approximations of the world where agents act. For instance, Logistics concerns moving packages among cities by airplanes and trucks; in the real world only a number of air routes are permitted, while in the standard PDDL Logistics domain airplanes can move between any pair of airports. This simplification could be adopted because, e.g., the exact network of the air routes is unknown to the domain model engineer.

For each of the considered domains we developed a simulator that simulates the physics of the domain with some discrepancies w.r.t. the available PDDL model. Differently from the Logistics domain model, a number of air and road routes are forbidden in the simulator. Specifically, all the airports but one are partitioned into two sets, each airplane can visit airports in only one of the two sets, plus a special airport that is not in either set. Similarly, all the locations within each city are divided into two overlapping sets, each truck can visit locations in only one of the two sets. We assume that GPS trackers are installed on board of both trucks and airplanes, RFID readers are installed on board of trucks as well as in storage areas, and that there are RFID tags stuck on packages. The simulation of an action of the Logistics domain outputs a perception consisting of readings made by GPS trackers and RFID readers. The GPS coordinates of a location are random numbers ranging from 1500 to 30,000; each reading made by the GPS tracker of a vehicle at a certain location is a pair of GPS coordinates corresponding to the location of the vehicle with a noise ranging from 0 to 5; the reading made by the RFID reader at location l for the RFID tag of package p is a random number ranging from 0.8 to 1, if p is at l; it is a random number ranging from 0 to 0.2 otherwise; the same reading is made by the RFID reader installed on board of a vehicle for the RFID tag of a package. For Logistics and all other tested domains, the noise is sampled according to a Gaussian distribution.

Domain Grid concerns moving a robot among a grid of rooms, some of which are closed by doors that can be opened by keys located in different rooms. A robot can move from room x to room y only if the two rooms are adjacent in the grid. Differently from the (standard PDDL) Grid domain, in the simulator x and y need to be connected in order for the robot to move between the two adjacent rooms, and only 3 over 4 adjacent rooms are connected. We assume that a GPS tracker and a RFID reader are installed on board of the robot, keys have GPS trackers and RFID tags, and there are sensors mounted on doors which detect whether doors are open or closed. The simulation of an action of the Grid domain outputs a perception consisting of readings made by GPS trackers, RFID readers, and door sensors. The GPS coordinates of room (x, y) in the grid are $(100 \cdot x, 100 \cdot y)$. The reading made by the GPS tracker of the robot at a certain room is a pair of GPS coordinates corresponding to the room coordinates with a noise ranging from 0 to 5; the same reading is done by the GPS trackers mounted on keys. The reading made by the RFID reader of the robot for the RFID tag of a certain key is a random number ranging from 0.8 to 1.0, if the key is grasped by the robot; it is a random number ranging from 0 to 0.2 otherwise. Similarly, the reading of the door sensors is a random number ranging from 0.8 to 1.0, if the door is open; it is a random number ranging from 0 to 0.2, if the door is closed.

Domain Rovers concerns moving rovers on the surface of a planet, taking images, collecting samples, and communicating images back to a lander. A rover at a waypoint can take an image of an objective only if the objective is visible from the waypoint. Similarly, a rover at a waypoint can communicate back data to the lander only if the lander is visible from the waypoint. Differently from the PDDL domain model, the simulator does not allow to take image at a half of the waypoints from which an objectives is visible, and it does not allow to communicate back data to the lander at a half of the waypoints from which the lander is visible.

Table 1: Minimum and maximum number of domain states (1st column), actions (2nd column), perception variables (3rd column), and number of states learned by PAL with the CONTINUE setting (4th column) over the instances of our benchmark domains solved by PAL.

Domain	S	A	PV	LS
Logistics	[e+21, e+219]	[650, 151400]	[269, 18152]	[856, 4864]
Grid	[e+07, e+35]	[726, 26135]	[47, 147]	[204, 1983]
Rovers	[e+10, e+68]	[362, 33732]	[165, 3961]	[114, 1286]

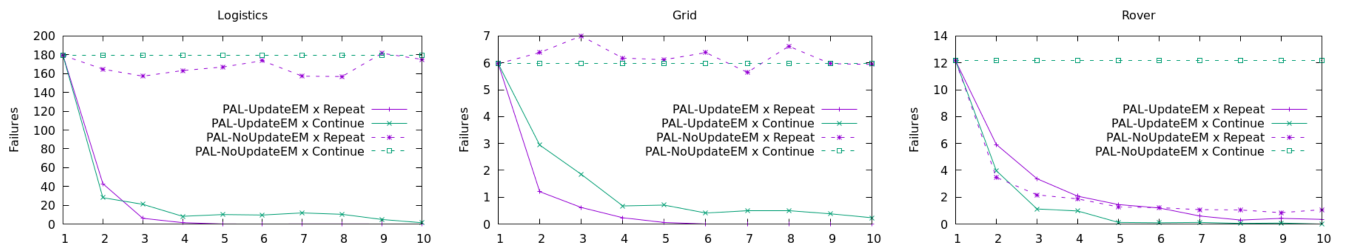

Figure 2: Average number of action-execution failures of PAL-NoUpdateEM and PAL-UpdateEM with settings REPEAT and CONTINUE for 10 episodes of domains Logistics, Grid, and Rovers.

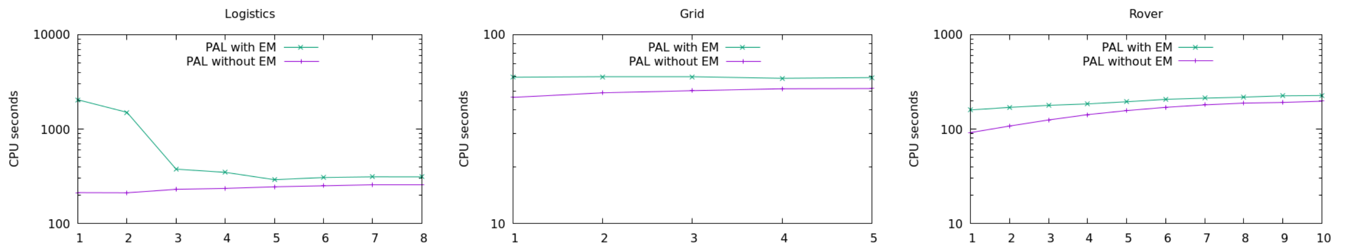

Figure 3: Average CPU-time of PAL using the REPEAT setting with/without the exploration model (EM) for up to 10 episodes of domains Logistics, Grid, and Rovers.

We assume that rovers have GPS trackers on board, and that there are sensors which output real numbers on the basis of the truth values of facts of the domain. The simulation of an action of domain Rovers outputs a perception consisting of readings made by GPS trackers and sensors. The GPS coordinates of a waypoint are random numbers ranging from 0 to 3400; the reading made by the GPS tracker of a rover at a waypoint are the same GPS coordinates as for the waypoint with a noise ranging from 0 to 5; the reading of the sensors is a random number ranging from 0.8 to 1.0, if the fact the sensor detects is true, it is a random number ranging from 0 to 0.2 otherwise.

We generated and tested the following PAL problems: 37 problems derived from the largest instances of Logistics used in the first two International Planning Competitions (IPCs) (Bacchus 2001; McDermott 2000); the 5 problems derived from the instances of Grid used in the first IPC (McDermott 2000) plus 30 problems derived from randomly generated instances; and 40 problems derived from the instances of Rovers used in the third IPC (Fox and Long 2011). The initial and goal perceptions of the PAL problems were derived from the initial states and sets of goals of the relative IPC problems.

Experimental results The first experiment we conducted is running PAL with the IPC version of the planning domain for the input exploration model, and empty models for the input extensional and perception models. Algorithm 1 updates the exploration model when the execution of an action a in a state fails so that, when the exploration planner is run again from this state, the first action to execute in the new plan is different from a. For simplicity, in the current implementation of the (automatic) revision of the exploration model, if the execution of an action fails in a state, the model is modified in a way that such an action is never executable. In PDDL, we do this by adding auxiliary predicates and action preconditions that constraint the grounding of an operator to generate only actions that are not forbidden.

Even if the physics of the world encoded in the exploration model and in the simulator have discrepancies (as previously described), PAL using FastDownward can solve 30 over 37 instances of Logistics, and all the instances of Grid and Rovers. Table 1 shows that PAL using the exploration model is able to solve quite large problems. On the contrary, PAL without an exploration model solves no problem of our benchmarks, because it explores the world states randomly and only tiny problems where goals are "accidentally" reached can be solved.

Then, we tested PAL with non-empty input extensional and perception models. For each PAL problem, we repeatedly ran PAL with two different settings. In the first settings, PAL is run with the same initial and goal perceptions as those of the PAL problem. We call each of these run an *episode*. In the second setting, for each PAL problem we constructed a set $\mathbf{X}_g$ of goal perceptions derived from randomly generated sets of PDDL goals. For the first episode, PAL is run with the same initial and goal perceptions as those of the PAL problem; for each other episode, PAL is run with the last perception sensed in the previous episode as initial perception and a perception among those in $\mathbf{X}_g$ as goal perception. Essentially, for this second setting PAL continues to plan for incoming goals. In the following, the first and second settings are called REPEAT and CONTINUE, respectively.

We considered ten episodes and two versions of PAL. For both versions the input knowledge is the same but the exploration models are different. For every episode except the first one, the extensional and perception models are those derived

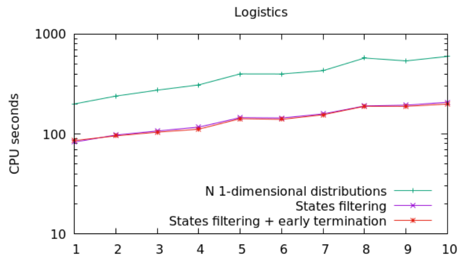

Figure 4: Average CPU-time of PAL with the CONTINUE setting using five different methods for determining the next search state for 10 episodes of domain Logistics.

at the end of the previous episode; for the first episode they are empty. One of the two versions of PAL has in input the IPC domain model as exploration model, the other version has in input the exploration model derived at the end of the previous episode. We denote these two versions of PAL with PAL-NoUpdateEM and PAL-UpdateEM, respectively.

Figure 2 shows the number of action-execution failures of PAL-NoUpdateEM and PAL-UpdateEM for settings REPEAT and CONTINUE. Since the exploration model is an approximation of the real world, the execution (through the simulator) of an action in the plan computed by the exploration planner can fail. The results in Figure 2 shows that for both settings REPEAT and CONTINUE the number of failures when the exploration model is updated is significantly lower than when no update is done among the episodes. For the REPEAT setting and every considered domain the number of failures reduces nearly to zero at the fifth episode. Remarkably, even for the CONTINUE setting the number of failures tends to decrease and is close to zero after few episodes. This is because the number of not executable actions is incrementally learned starting from the first episode, which reduces the chance of action failures among episodes.

When the agent's goals are satisfied in a state previously reached (and learned) by the agent, using the extensional planner can provide great computational benefits. This is because, typically, the (learned) state space searched by the extensional planner is much smaller than the state space induced by the exploration model searched by the exploration planner. To evaluate these benefits, we conducted an experiment using the REPEAT setting. First, we run PAL using the exploration planner; then, we run PAL without using the exploration planner but having the extensional and perception models learned by PAL in the first run (that used the exploration planner). Figure 3 shows the average CPU time of PAL with/without the exploration planner for up to ten episodes. For all the instances and episodes greater than 8 in Logistics, and greater than 5 in Grid, we have no action-execution failure for PAL using the exploration planner; hence the performance gap is the same as for the last episode shown in the figure. As expected, the CPU time of PAL using only the extensional planner is lower than when using the exploration planner. However, achieving the goal by the extensional planner is still, somewhat surprisingly, quite expensive. This is because determining the next current state of the agent from the sensed perception can be computationally much expensive when the number of perception variables and (learned) states in the extensional model is high.

To determine the next state, we filter the set of previously reached (learned) states according to the set J of perception variables that have been significantly changed by the execution of the last action. The next state is selected from the set of states satisfying $\rho_j(x_j \mid s) > t_j$ for each perception $j \in J$ according to the max-likelihood criteria. If this set is empty, then a new state is introduced. In Figure 4, such a strategy is denoted by *"State filtering"*. Finally, we consider another strategy. If the perception $\mathbf{x}$ is obtained by executing action a in state s and the extensional model contains the transition (s, a, s'), then, if for each perception variable i the likelihood of sensing x_i being in s' is above threshold t_i, the next state is s'. We run PAL using such a strategy together with the strategy for filtering states; this version of PAL is denoted by *"State filtering + early termination"*. The results in Figure 4 show that, on average, for Logistics the filtering of states significantly improves the performance, while the speedup obtained using the early termination is negligible.

However, determining the next state can be a bottleneck of the approach even using these strategies. Figure 5 shows, for the CONTINUE setting, the average CPU time required by PAL for planning and for determining the next state w.r.t. the average total CPU time. For the last episodes, determining the next state is more time consuming that using the exploration planner. This is because, for the last episodes, (i) the number of action-execution failures is low or zero, and hence the number of times the exploration planner is run is also low; (ii) the computational cost required to determine the next states increases with the number of visited states, which progressively increases with the episodes.

6 Related Work

A large amount of work on learning planning domains focuses on learning action schema from data. Gregory and Cresswell (2016), McCluskey et al. (2009), and Cresswell, McCluskey, and West (2013) propose learning general action schema in a structured language starting from plans containing grounded application instances of actions. Mourão et al. (2012) learn action schemata from noisy and incomplete observations. Each observation is a sequence of alternating actions and set of fluent expressions. Zhuo and Yang (2014) learn an action schema on a target domain by transfer learning from a set of source domains and by observing partial plan traces. Aineto, Jiménez, and Onaindia (2018) propose a method for learning action models from observations of plan executions that compiles the learning task into a classical planning task. In all these approaches, learning is performed at the symbolic level, and mappings to perceptions in a continuous environment are not considered. This is also the case of the work by Bonet and Geffner (2019), which provides a framework for learning first-order symbolic representations from plain graphs. Indeed, plain graphs are state transition systems, and there is no mapping to perceptions in a continuous space, which is what sensors actually provide. All the above mentioned works do not tackle the problem of finding an abstraction of the continuous environment (with

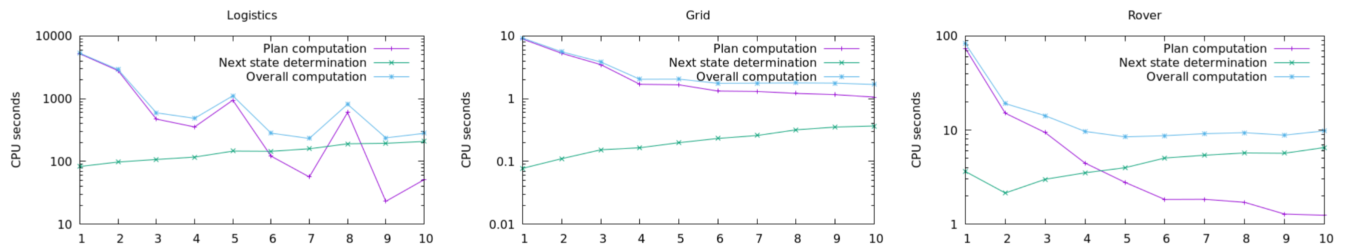

Figure 5: Average CPU-time for planning, determining the next search state, and total time required by PAL with the CONTINUE setting for each episode among 10 episodes of domains Logistics, Grid, and Rovers.

continuous states) into a finite set of states. Building this abstraction (encoded in the perception function) is one of the key contributions of the PAL framework.

There are however a set of approaches that learn a discrete planning domain from a continous environment. Causal InfoGAN learns discrete or continuous models from high dimensional sequential observations (Kurutach et al. 2018). This approach fixes a priori the size of the discrete domain model. Differently from our approach, their goal is to generate an execution trace in the high dimensional space. LatPlan takes in input pairs of high dimensional raw data (e.g., images) corresponding to transitions (Asai and Fukunaga 2018; Asai 2019). LatPlan is an offline approach, while our approach is online and works also in dynamic environments. Konidaris, Kaelbling, and Lozano-Pérez (2018) construct a STRIPS model by learning the Boolean atoms of the preconditions and effects of actions. However, their basic assumption is that a continuous model of the world is available, and that it is possible to know a fixed mapping from the continuous model to the deterministic classical planning domain. We do not rely on such assumptions. Moreover, in our approach, the mapping through perception functions is learned dynamically. Finally, in the work by Konidaris, Kaelbling, and Lozano-Pérez (2018), the mapping is set-theoretic, while we allow for a probabilistic mapping through a probability density function.

Most of the work on learning and planning in Partially Observable Markov Decision Processes (POMDP) – see, e.g., (Ross et al. 2011; Katt, Oliehoek, and Amato 2017) – focuses on learning transitions and policies by assuming a fixed and given set of states and a given reward function. Some of them drop the assumption of a bounded state space, see, e.g., (Doshi-Velez 2009). However, none of these works uses an intensional representation to guide the search for learning an extensional representation of the planning domain.

Our approach shares some similarities with the work on planning by reinforcement learning (Kaelbling, Littman, and Moore 1996; Sutton and Barto 1998; Geffner and Bonet 2013; Yang et al. 2018; Parr and Russell 1997; Ryan 2002; Leonetti, Iocchi, and Stone 2016; Garnelo, Arulkumaran, and Shanahan 2016), since we learn by acting in the environment. However, these works focus on learning policies and assume the set of states and the correspondence between continuous data from sensors and states are fixed.

A complementary approach is pursued in works that plan and learn directly in a continuous space, see e.g., (Abbeel, Quigley, and Ng 2006; Mnih et al. 2015; Co-Reyes et al. 2018). These approaches do not require a perception function, since there is no abstract discrete model of the world. Such approaches are very suited to address some tasks, e.g., moving a robot arm to a desired position or performing some manipulations. However, we believe that, in several situations, it is conceptually appropriate and practically efficient to learn an abstract discrete and deterministic model where planning is much easier and efficient to perform.

We share the idea of a planning domain at the abstract level with all the work on abstraction on MDP models, see, e.g., (Abel et al. 2018; Li, Walsh, and Littman 2006). However, our problem and approach is substantially different, since in the work about abstraction on MDP models, the mapping between original MDP states and abstract states is given, while we learn it.

7 Conclusions

This paper extends the approach proposed in (Serafini and Traverso 2019a; 2019b) by allowing the agent to exploit some prior belief on the environment expressed in PDDL. This is a believed model not a correct one. This exploration model guides the learning of the extensional representation of the planning domain to achieve a given goal. We have introduced a novel algorithm that (i) interleaves symbolic planning in the intensional representation (the PDDL model) and searching in the extensional representation (the state transition system); (ii) implements a goal directed heuristic to achieve a given goal; (iii) incrementally updates the intensional representation by collecting information about actions that are not executable. Moreover, we have proposed a new formulation of the perception function that overcomes the limits of the definition given by Serafini and Traverso (2019b) for determining the next state when the number of states and perception variables is high. Finally, we have experimentally shown that the novel approach can scale up to large state spaces, while the the previous approach (Serafini and Traverso 2019a; 2019b) could deal with only very small state spaces.

References

Abbeel, P.; Quigley, M.; and Ng, A. Y. 2006. Using inaccurate models in reinforcement learning. In *ICML*.

Abel, D.; Arumugam, D.; Lehnert, L.; and Littman, M. L. 2018. State abstractions for lifelong reinforcement learning. In *ICML*.

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

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI*.

Asai, M. 2019. Unsupervised grounding of plannable first-order logic representation from images. In *ICAPS*.

Bacchus, F. 2001. The AIPS '00 planning competition. *AI Magazine* 22(3):47–56.

Bonet, B., and Geffner, H. 2019. Learning first-order symbolic planning representations from plain graphs. *CoRR* abs/1909.05546.

Co-Reyes, J. D.; Liu, Y.; Gupta, A.; Eysenbach, B.; Abbeel, P.; and Levine, S. 2018. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *ICML 2018*.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using *LOCM*. *Knowledge Eng. Review* 28(2):195–213.

Doshi-Velez, F. 2009. The infinite partially observable markov decision process. In *NIPS*, 477–485.

Fikes, R., 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.

Fox, M., and Long, D. 2011. The 3rd international planning competition: Results and analysis. *CoRR* abs/1106.5998.

Garnelo, M.; Arulkumaran, K.; and Shanahan, M. 2016. Towards deep symbolic reinforcement learning. *CoRR* abs/1609.05518.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning - Theory and Practice*. Elsevier.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Gregory, P., and Cresswell, S. 2016. Domain model acquisition in the presence of static relations in the LOP system. In *IJCAI*.

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

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *J. Artif. Intell. Res.* 4:237–285.

Katt, S.; Oliehoek, F. A.; and Amato, C. 2017. Learning in pomdps with monte carlo tree search. In *ICML*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Pérez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *J. Artif. Intell. Res.* 61:215–289.

Kurutach, H.; Tamar, A.; Yang, G.; Russell, S.; and Abbeel, P. 2018. Learning plannable representations with causal infogan. In *NIPS*.

Leonetti, M.; Iocchi, L.; and Stone, P. 2016. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artif. Intell.* 241:103–130.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for mdps. In *ISAIM*.

McCluskey, T. L.; Cresswell, S.; Richardson, N. E.; and West, M. M. 2009. Automated acquisition of action knowledge. In *ICAART*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C. A.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. E. 1998. PDDL—The planning domain definition language. Technical Report DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, Connecticut.

McDermott, D. V. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Mourão, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS operators from noisy and incomplete observations. In *UAI*.

Parr, R., and Russell, S. J. 1997. Reinforcement learning with hierarchies of machines. In *NIPS*.

Ross, S.; Pineau, J.; Chaib-draa, B.; and Kreitmann, P. 2011. A bayesian approach for learning and planning in partially observable markov decision processes. *Journal of Machine Learning Research* 12:1729–1770.

Ryan, M. R. K. 2002. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In *ICML*, 522–529.

Serafini, L., and Traverso, P. 2019a. Incremental learning of discrete planning domains from continuous perceptions. *CoRR* abs/1903.05937.

Serafini, L., and Traverso, P. 2019b. Learning abstract planning domains and mappings to real world perceptions. In *AI*IA 2019 - Advances in Artificial Intelligence*, volume 11946 of *Lecture Notes in Computer Science*, 461–476. Springer.

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

Yang, F.; Lyu, D.; Liu, B.; and Gustafson, S. 2018. PEORL: integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In *IJCAI*.

Zhuo, H. H., and Yang, Q. 2014. Action-model acquisition for planning via transfer learning. *Artif. Intell.* 212:80–103.