One-Shot Learning of Temporal Features on Action Models via Constraint Programming

Antonio Garrido and Sergio Jiménez

Abstract—This work proposes a novel constraint programming approach for learning the temporal features of durative actions in an expressive temporal planning model with overlapping actions, which makes it suitable for learning in multi-agent environments. We analyze the extreme scenario, where just a single (one-shot) partial observation of the execution of a temporal plan is available, to learn the distribution of conditions/effects and estimate the durations, resulting in a consistent constraint model. Our approach automatically builds a purely declarative formulation that models time-stamps for actions, causal link relationships (conditions and effects), threats and effect interferences that appear in planning. It also accommodates a different range of expressiveness, subsuming the PDDL2.1 temporal semantics. Our formulation is simple but effective, and is not only valid for learning, but also for plan validation, as shown in its evaluation that returns high precision and success ratios. Finally, our formulation is solver-independent, meaning that an arbitrary CSP solver can be used for its resolution.

1 Introduction

Automated planning is the model-based approach for the task of selecting the actions that achieve a given set of goals starting from a given initial state. *Classical planning* (*aka* STRIPS planning) is the vanilla model for planning and it assumes: fully observable states under a deterministic world, instantaneous actions, and goals that are exclusively referred to the last state reached by a plan [1], [2]. Beyond classical planning, there is a bunch of more expressive planning models that relax the previous assumptions to compute more detailed solutions than classical plans [2].

Temporal planning is one of these more expressive planning models, as it relaxes the assumption of instantaneous actions [3]. Temporal actions have durations and conditions/effects that must hold/happen at different times, which means that temporal actions can be executed in parallel and overlap in several ways [4]. Consequently, valid solution plans for temporal planning problems need to indicate the precise time-stamp when an action starts and ends [5].

1.1 Motivation

Despite the potential of state-of-the-art planners, its applicability to the real world is still somewhat limited because of the difficulty and time-consuming of manually specifying correct and complete planning models [6], [7], particularly in expressive temporal action languages [4]. The more expressive the planning model is, the more evident becomes this knowledge engineering bottleneck, which jeopardizes the usability of AI planning technology. This has led to a growing interest in the planning community for the learning of action models, as a cognitive task [8], to improve their quality and reduce the human effort [7], [9]. The objective of this learning task is to understand and predict action models that are *consistent* with a set of noiseless observations (defined as some sequence of state changes, input constraints, world transitions, expert demonstrations or plan traces/logs). Model learning from observation of past behavior provides indirect, but very valuable, information to hypothesize the action models, thus helping future planning decisions and recommendations of preference models [10]. This is specially interesting for proactive assistants when recognizing activities of multiple (human or software) agents to assist them in their daily activities and, as the ultimate frontier, to generate personalized recommendations, predict and anticipate their needs or actions.

Most approaches for learning planning action models are purely inductive and often require large datasets of observations, e.g. thousands of plan observations, to compute a statistically significant model that minimizes some error metric over the observations [7], [11], [12], [13], [14]. Defining model learning as an optimization task over a large set of observations does not guarantee completeness (the learned model may fail to explain an observation), nor correctness (the states induced by the execution of the plan generated with the model may contain contradictory information). Further, in many real-world applications it is expensive to collect large datasets of observations for training [15], or even impossible. This often happens when the number of samples is limited or when a human needs to perform repetitive actions (learning by demonstration). Therefore, reducing the effort to deal with tractable datasets is desirable [15].

Learning an action model for classical planning involves computing the actions' conditions and effects that are consistent with the input observations. Learning classical action models has been addressed by different approaches [16]. Since pioneering learning systems like ARMS [13], we have seen systems able to learn STRIPS action models with quantifiers [7], [17], from noisy actions or states [12], [14],

• *A. Garrido and S. Jiménez are with VRAIN, Valencian Research Institute for Artificial Intelligence. Universitat Politecnica de Valencia, Camino de Vera s/n, 46022. Valencia (Spain).*
E-mail: {agarridot,serjice}@dsic.upv.es

from null state information [18], from incomplete domain models [19], [20], or from partially observed plan traces [21]. But, to our knowledge, none of these systems deals with durative actions to learn their temporal features. Learning the temporal features means: i) to identify how actions' conditions and effects are temporally distributed in the action execution; ii) to estimate the action duration according to the observations; and, after all, iii) to model a class of highly expressive durative actions that meet the challenges of real-world planning. Roughly speaking, learning a classical action model aims to decide which are the conditions/effects, whereas learning the temporal features aims to decide when and how they are used. And these two learning tasks are asymptotically very complex.

As a motivating example, let us assume a logistics scenario. When driving a truck between two locations, the road should be available all over the execution of the action. However, if we use a plane for transportation, we will only need an airport at the beginning+end of the action. On the other hand, given a full logistics trace, we can deduce the precise duration of an action. But if observations are partial, we might only observe when the trip starts, but not the exact time when it ends (traffic congestion may affect). Or, analogously, we might know when the trip ends but be unsure about when it started. In classical planning none of the previous statements entails difficulty: the distribution of conditions/effects is fixed, as they are always allocated at the start/end of the action, respectively, and all actions have unitary duration, which is not very realistic. In temporal planning, if a human modeler perfectly knows the semantics of the logistics actions, learning the temporal features may not be a complicated task either. But, without that expert knowledge, if the human has to distribute conditions arbitrarily named $\{p, q, r\}$, effects $\{not-p, not-q, s\}$ and estimate valid durations for unknown actions $\{a1, a2, a3\}$ over a set of partial observations, the task becomes much more complex. The learning task induces many assignments that need to fit a complex puzzle, because not all combinations are valid and they depend on the interactions with other actions, observations and constraints. And surely this is not easy for a human. Eventually, learning the temporal features will allow us: i) to better understand the insights of the logistics in terms of what is possible (or not) and why, because the model is consistent with the observed data; ii) to suggest changes that can improve the model originally created by a human, e.g. re-distributing the actions' conditions, provided they still explain the observations; and iii) to automatically elaborate similar models for similar scenarios (i.e. knowledge transfer or transfer learning [15]), such as public transit for commuters, tourists or people in general in metropolitan areas —*aka* smart urban mobility.

1.2 Proposed Work

Based on a classical model of actions, this paper analyzes a novel application of Constraint Programming (CP) for the *one-shot learning* of the temporal features; that is, the extreme case of learning from a single and partially specified model observed from the execution of a temporal plan. On the one hand, our main contribution in the field of knowledge engineering is a constraint formulation for a very expressive temporal planning model that follows the work in [22]. Such a work proposes a CP formulation, automatically generated from a given model of actions, to be used for planning and/or scheduling a whole plan. We keep the philosophy of using CP but, contrary to [22], we address the inverse task now: learn the temporal features of an action model given a plan trace and a partial model of actions. Intuitively, the input model of actions used in [22] is our output now, as we are interested in learning/playing the designer's role, *w.r.t* the temporal features, rather than in planning. On the other hand, another contribution is to learn as much knowledge as possible from the lowest amount of observations. Rather than using a machine learning approach that builds a statistical model that requires huge datasets, we exploit CP inference for deductive reasoning on arbitrary size plan traces.

In our work we focus on the temporal aspects and rely on the classical conditions+effects. Although this could seem a limitation of our proposal, there is no need to deliberately learn them again as they can be obtained by existing approaches, previously modeled by a human, or observed. But (partial) observations may now refer to the execution of overlapping durative actions, in opposition to most of the existing approaches that work with sequential non-durative actions. Working with overlapping makes our approach suitable for learning in multi-agent environments, which is also a contribution to deal with realistic scenarios. Finally, the paper evidences that the learning of temporal features strongly resembles the task of synthesizing and validating a plan that satisfies all the imposed constraints or, in other words, that is consistent with the noiseless input observations. This is our last contribution: a unified CP formulation for both learning and plan validation, which supports temporal models and addresses plan validation of partial action models beyond the functionality of standard validators [5].

The paper is organized as follows. Section 2 introduces the background and terminology used. Section 3 formalizes the learning task of temporal features, and exemplifies the difficulty of such learning and the huge number of combinations that appear. In Section 4 we elaborate our CP formulation for learning, presenting the variables, constraints and some simple heuristics. It also shows how our formulation is flexible enough to be used for plan validation. The evaluation and experimental results are shown in Section 5. Finally, Section 6 concludes the paper and proposes some future work.

2 Background and Terminology

This section formalizes the *classical* and *temporal* planning models that we follow in this work, as well as the *Constraint Satisfaction Problem*.

2.1 Classical Planning

Let F be a set of facts that represent propositional variables. Without loss of generality, we assume that F does not contain conflicting values f and $not-f$. A state s is a full assignment of boolean values to variables, $|s| = |F|$, so the size of the state space is $2^{|F|}$. A *classical planning problem* is

```
(:action board-truck
 :parameters (?d - driver ?t - truck ?l - location)
 :precondition (and (at ?d ?l) (empty ?t) (at ?t ?l))
 :effect (and (not (at ?d ?l)) (not (empty ?t))
              (driving ?d ?t)))

(:action drive-truck
 :parameters (?t - truck ?from - location ?to - location
              ?d - driver)
 :precondition (and (at ?t ?from) (link ?from ?to)
                    (driving ?d ?t))
 :effect (and (not (at ?t ?from)) (at ?t ?to)))
```

Fig. 1. PDDL schema for two classical actions from the *driverlog* domain.

a tuple $\langle F, I, G, A\rangle$, where I is the initial state, $G \subseteq F$ is a set of goal conditions over F, and A is the set of actions that modify states. We assume that actions are grounded from action schemas or operators, as in PDDL (Planning Domain Definition Language [2], [3]). A grounded action is a schema where all the parameters are fully instantiated.

Each action $a \in A$ has a set of preconditions $\mathsf{pre}(a)$ and a set of effects $\mathsf{eff}(a) = \{\mathsf{eff}^+(a) \cup \mathsf{eff}^-(a)\}$; $\mathsf{pre}(a), \mathsf{eff}^+(a), \mathsf{eff}^-(a) \subseteq F$. $\mathsf{pre}(a)$ must hold before a starts (this is why they are named *pre*conditions), whereas $\mathsf{eff}(a)$ happen when a ends. This way, a is applicable in a state s if $\mathsf{pre}(a) \subseteq s$. When a is executed, a new state, the successor of s, is created that results of applying $\mathsf{eff}(a)$ on s. Typically, $\mathsf{eff}(a)$ is formed by positive and negative/delete effects ($\mathsf{eff}^+(a)$ and $\mathsf{eff}^-(a)$, which are asserted and retracted, respectively). Fig. 1 shows an example of two classical actions for a logistics scenario, from the *driverlog* domain of the International Planning Competition (IPC).[1] Action `board-truck` boards a driver on an empty truck at a given location. In `drive-truck` a truck is driven between two locations, provided there is a link between them.

In this work we define a plan for a classical planning problem as a set of pairs $\langle (a_1, t_1), (a_2, t_2) \dots (a_n, t_n)\rangle$. Each (a_i, t_i) pair contains an instantaneous action a_i and the planning step t_i when a_i starts. This action sequence induces a state sequence $\langle s_1, s_2 \dots s_n\rangle$, where each a_i is applicable in s_{i-1}, being $s_0 = I$, and generates state s_i. In every valid plan $G \subseteq s_n$, i.e. the goal condition is satisfied in the last state. In general terms, classical plans can be sequential plans, where only one action is executed at each planning step, or parallel plans, where several actions can be executed at the same planning step.

2.2 Temporal Planning

A *temporal planning problem* is also a tuple $\langle F, I, G, A\rangle$ where F, I and G are defined like in classical planning, and A represents the set of *durative actions*. There are several options that allow for a high expressiveness of durative actions. On the one hand, an action can have a fixed duration, a duration that ranges within an interval or a distribution of durations. On the other hand, actions may have conditions/effects annotated at different times, such as conditions that must hold some time before the action starts, effects that happen just when the action starts, in the middle of the action or some time after the action finishes [22].

1. www.icaps-conference.org/index.php/Main/Competitions

```
(:durative-action board-truck
 :parameters (?d - driver ?t - truck ?l - location)
 :duration (= ?duration 2)
 :condition (and (at start (at ?d ?l))
                 (at start (empty ?t))
                 (over all (at ?t ?l)))
 :effect (and (at start (not (at ?d ?l)))
              (at start (not (empty ?t)))
              (at end (driving ?d ?t))))

(:durative-action drive-truck
 :parameters (?t - truck ?from - location ?to - location
              ?d - driver)
 :duration (= ?duration (driving-time ?from ?to))
 :condition (and (at start (at ?t ?from))
                 (at start (link ?from ?to))
                 (over all (driving ?d ?t)))
 :effect (and (at start (not (at ?t ?from)))
              (at end (at ?t ?to))))
```

Fig. 2. PDDL2.1 schema for two durative actions from the *driverlog* domain.

A popular model for temporal planning is given by PDDL2.1 [3], a language that somewhat restricts temporal expressiveness, which defines a durative action a with the following elements:

- $\mathsf{dur}(a)$, a positive value for the action duration.
- $\mathsf{cond}_s(a), \mathsf{cond}_o(a), \mathsf{cond}_e(a)$. Unlike the *pre*conditions of a classical action, now conditions must hold before a (*at start*), during the entire execution of a (*over all*) or when a finishes (*at end*), respectively. In the simplest case, $\mathsf{cond}_s(a) \cup \mathsf{cond}_o(a) \cup \mathsf{cond}_e(a) = \mathsf{pre}(a)$.[2]
- $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$. Now effects can happen *at start* or *at end* of a, respectively, and can still be positive or negative. Again, in the simplest case $\mathsf{eff}_s(a) \cup \mathsf{eff}_e(a) = \mathsf{eff}(a)$.[3]

The semantics of a PDDL2.1 durative action a can be defined in terms of two discrete events, $\mathsf{start}(a)$ and $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. This means that if action a starts on state s, $\mathsf{cond}_s(a)$ must hold in s; and ending a in state s' means $\mathsf{cond}_e(a)$ holds in s'. *Over all* conditions must hold at any state between s and s' or, in other words, throughout interval $[\mathsf{start}(a)..\mathsf{end}(a)]$. Analogously, *at start* and *at end* effects are instantaneously applied at states s and s', respectively —continuous effects are not considered. Fig. 2 shows two durative actions that extend the classical actions of Fig. 1. Now `board-truck` has a fixed duration whereas in `drive-truck` the duration depends on the two locations.

PDDL2.2 is an evolution of PDDL2.1 that extends the initial state I with the notion of *Timed Initial Literals* [23] ($\mathsf{til}(f, t)$ or $\mathsf{til}(not - f, t)$). A TIL is introduced as a way of asserting or retracting a fact $f \in F$ at a certain time t, independently of the actions in the plan.[4] TILs are very useful

2. In classical planning, $\mathsf{pre}(a) = \{p, not - p\}$ is contradictory. In temporal planning, $\mathsf{cond}_s(a) = \{p\}$ and $\mathsf{cond}_e(a) = \{not - p\}$ is a possible situation, though very unusual.

3. In classical planning, $\mathsf{eff}(a) = \{p, not - p\}$ is contradictory. In temporal planning, $\mathsf{eff}_s(a) = \{not - p\}$ and $\mathsf{eff}_e(a) = \{p\}$ is a possible and frequent situation to block/unblock a resource p used within a.

4. Intuitively, the information in I can be seen as a particular case of TILs with time $t = 0$.

in temporal planning to define exogenous happenings; for instance, a time window when a warehouse is open in a logistics scenario ($\mathsf{til}(open, 8)$ and $\mathsf{til}(not - open, 20)$).

A temporal plan is a set of pairs $\langle (a_1, t_1), (a_2, t_2) \dots (a_n, t_n) \rangle$. Each (a_i, t_i) pair contains a durative action a_i and $t_i = \mathsf{start}(a_i)$. This temporal plan induces a state sequence formed by the union of all states $\{s_{t_i}, s_{t_i + \mathsf{dur}(a_i)}\}$, where there exists a state $s_0 = I$, and $G \subseteq s_{end}$, being s_{end} the last state induced by the plan. Though a sequential temporal plan is syntactically possible, it is semantically useless. Consequently, temporal plans are always given as parallel plans, which is essential for multi-agent planning.

2.3 Constraint Satisfaction Problems

A *Constraint Satisfaction Problem* (CSP) is defined as the tuple $\langle V, D, C \rangle$, where:

- $V = \langle v_1, v_2 \dots v_n \rangle$ is a set of n finite domain variables.
- $D = \langle D_{v_1}, D_{v_2} \dots D_{v_n} \rangle$ are the respective domains defining the set of possible values for each variable $v_i \in V$.
- $C = \langle c_1, c_2 \dots c_m \rangle$ is a set of constraints binding the possible values of the variables in V. Every constraint $c_i \in C$ is defined as a pair $c_i = \langle V_i, R_i \rangle$, where:
  - $V_i \subseteq V$ is a subset of $k \leq n$ variables.
  - R_i is a k-ary relation on the corresponding subset of domains.

An evaluation ev_i satisfies a constraint $c_i = \langle V_i, R_i \rangle$ if the values of ev_i assigned to the variables in V_i satisfy the relation R_i. An evaluation of values to all variables in V is *consistent* if it does not violate any of the constraints in C, i.e. it is a solution for the CSP $\langle V, D, C \rangle$. Note that we deal with a satisfaction problem and, consequently, the CSP can have many solutions.

3 One-Shot Learning of Temporal Features

3.1 Learning Task

We define our one-shot learning task of temporal features in an action model as a tuple $\langle F, I, G, A?, O \rangle$, where:

- $\langle F, I, G, A? \rangle$ is a temporal planning problem in which actions are partially specified. Actions in $A?$ are those observed in the plan trace. They are partially specified because we do not know the exact structure in terms of distribution of conditions/effects nor the duration. In this work we assume that, for each action $a \in A?$, we only know $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$, as this information can be extracted from the classical version of the planning problem, from prior knowledge we have on the problem, or given by an expert.
- O is the sequence of observations, provided by a sensor or human observer, corresponding to a plan trace which contains the time when an action a in $A?$ starts ($\mathsf{start}(a)$) or ends ($\mathsf{end}(a)$). In a partial observation context, we might observe the start of the action and/or its end. If we observe both $\mathsf{start}(a)$ and $\mathsf{end}(a)$ we can easily calculate $\mathsf{dur}(a)$; otherwise $\mathsf{dur}(a)$ needs to be estimated. In our work, we assume only one in $\{\mathsf{start}(a), \mathsf{end}(a)\}$ is observed to minimize the number of required (partial) observations. In practice, sensors usually observe changing properties of the word and deduce whether a proposition is true or false at a given time. In order to simulate the observation of the execution of action a, we can use dummy propositions $\mathsf{is\_start}(a), \mathsf{is\_end}(a)$ with values true/false, representing whether the action starts or ends.

Note that our definition of learning task requires the minimal amount of input knowledge on the states: only the result of the plan execution is observed. More particularly, we do not observe any intermediate state in pursuit of minimizing the number of observations. We only require I, as a full state, and G, as a partial state of goals.[5] A solution to this learning task is a fully specified model of temporal actions $\mathcal{A}$, with all actions of $A?$, where the duration and distribution of conditions/effects is completely specified. In other words, for each action $a \in A?$, we have its equivalent version in $\mathcal{A}$ where we have learned $\mathsf{dur}(a)$, $\mathsf{cond}_s(a)$, $\mathsf{cond}_o(a)$, $\mathsf{cond}_e(a)$, $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$. Actions in $\mathcal{A}$ must be consistent with the partial specification given in $A?$, which means having exactly the same conditions and effects, starting and/or ending as observed in O, and inducing a temporal plan from I that satisfies G. Intuitively, $\mathcal{A}$ is a solution to the learning task if it explains all the observations (completeness) and its subjacent temporal model implies no contradictions in the states induced by their execution (correctness).

As can be noticed, the learned model $\mathcal{A}$ must satisfy all the constraints imposed by both the partially specified model $A?$ and the observations O. Thus, formulating a CSP that states all the constraints to address the learning task, and finding a solution that explains such formulation seems a very promising approach.

3.2 Example. Is Learning the Temporal Features a Simple Task?

Given a partially specified model of actions $A?$ and a set of observations O, learning the model $\mathcal{A}$ may seem, a priori, a straightforward task as it *just* implies to distribute the conditions+effects in time and estimate durations. However, this is untrue.

Let us suppose the example of Fig. 3, with observations on the start times, conditions and effects of actions. Clearly, $a3$ needs $a1$ to have r supported, which represents the causal link or dependency $\langle a1, r, a3 \rangle$. Let us imagine that r is in $\mathsf{cond}_s(a3)$. In such a case, if r is in $\mathsf{eff}_s(a1)$, $\mathsf{dur}(a1)$ is irrelevant to $a3$, but if r is in $\mathsf{eff}_e(a1)$, $\mathsf{dur}(a1)$ has to be lower or equal than 5 ($\mathsf{start}(a1) + \mathsf{dur}(a1) \leq \mathsf{start}(a3)$). On the contrary, if r is in $\mathsf{cond}_e(a3)$, $\mathsf{dur}(a1)$ could be much longer. Therefore, the distribution of conditions and effects has a significant impact in the durations, and vice versa.

5. Other learning tasks can also include intermediate states and input knowledge on static information, which makes the learning task easier to be solved [24].

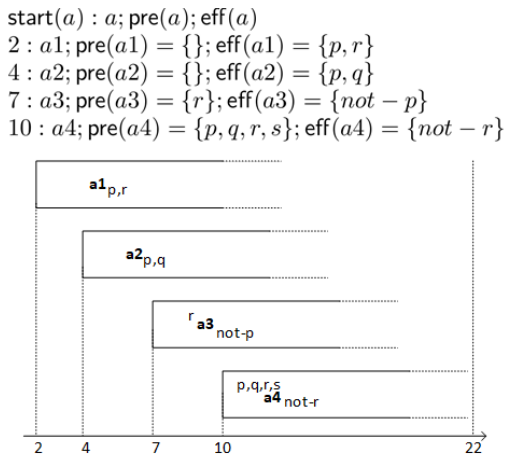

Fig. 3. A simple example of how learning the temporal features from O and $A?$ is not straightforward. We (optionally) observe the plan makespan is 22.

$a4$ needs p, which means two possible causal links ($\langle a1, p, a4\rangle$ or $\langle a2, p, a4\rangle$), but this depends on the effects+durations of $a1$ and $a2$. Therefore, the causal links are unknown, not easy to detect and they affect the structure of the temporal plan. But $a4$ really needs both $a1$ and $a2$ to have p, q, r supported. Let us imagine that p, q, r are in $\mathsf{cond}_s(a4)$ and p, q in $\mathsf{eff}_e(a2)$; then $\mathsf{dur}(a2) \leq 6$. Even if we knew for sure that $\mathsf{dur}(a2) = 6$ and r was in $\mathsf{eff}_e(a1)$, we could never estimate the exact value of $\mathsf{dur}(a1)$, as any value in $]0..8]$ would be valid. Intuitively, an action has to wait until the last of its supports, but we cannot grant when the previous supports happen. Therefore, in some situations the precise duration cannot be found and we can only provide values that make the model consistent. An analogous situation would happen if we only observed the end times and all the effects were *at start*.

On the other hand, $a3$ deletes p, which means that it might *threat* the causal link $\langle a1, p, a4\rangle$ or $\langle a2, p, a4\rangle$. But again, this threat depends on the distribution of conditions+effects and the durations. For instance, if $not - p$ is in $\mathsf{eff}_s(a3)$, then $a1$ or $a2$ must support p after time 7 and before $a4$ requires it, which entails many consistent alternatives. On the contrary, if p is in both $\mathsf{eff}_s(a1)$ and $\mathsf{eff}_s(a2)$, the observations on this plan trace are inconsistent as $a3$ deletes p and no other action in the plan supports p for $a4$ (no solution exists). However, if $not - p$ is in $\mathsf{eff}_e(a3)$, $\mathsf{dur}(a3) > 3$ and p is in $\mathsf{cond}_s(a4)$, then no threat will occur in the plan. Therefore, causal links and threats can easily appear or disappear depending on the selected distributions and durations.

Finally, there are some philosophical questions without a clearly motivated answer. First, why some conditions are modeled as *at start* and others as *over all*? In `drive-truck` of Fig. 2, why `(driving ?d ?t)` is required throughout the entire action but `(link ?from ?to)` only at its beginning? Apparently, the link between the two locations should remain all over the driving, which is more sensible. So is this a wrong decision of the human modeler? In Fig. 3, s is a condition that is not asserted nor deleted in the plan so it can be considered as static information, i.e. invariant (*over all*) knowledge that always holds.[6] Second, why some effects are modeled as *at start* and others as *at end*? In `board-truck`, why is `(not (empty ?t))` happening *at start* and `(driving ?d ?t)` *at end*? Could it be in the opposite way? Although this seems sensible here, in some domains it is modeled in the other way round. Third, what happens if one action requires/supports what it deletes (see $a4$ in Fig. 3, which might threat itself)? In such a case, the delete effect should happen later than its requirement/supporting. Four, what happens if all effects are *at start*? This makes little sense, as the duration of the actions would be undetermined and could potentially exceed the known plan horizon or makespan, no matter the problem goals. However, this is possible. In Fig. 3, if the effects of $a1$ and $a2$ are *at start*, is it sensible to allow their durations to pass a hypothetical limit of 22? In other words, once all plan goals are achieved, can the actions be executed beyond the plan makespan, or do they need to be cut off to such a value? This could potentially lead to an infinite number of models and overlapping situations. Although infrequent, some humans model durative actions only with start effects, which is a bit irrational. Also, temporally expressive examples with required concurrency [4], where envelope actions are essential to find a plan, significantly increases the number of overlapping situations.

As can be noticed, learning the temporal features is not a simple task whatsoever, and many possible combinations are feasible provided they fit the constraints the problem and observations impose. Therefore, formulating a CSP seems a promising approach to address this learning task.

4 A CP Formulation to Learn Temporal Features on Action Models

Our approach is to create a CSP that includes all the constraints the learning task requires. This includes: i) the observations of start and/or end times; ii) the actions' conditions, effects and durations; iii) the causal structure of the plan with all possible supports; and iv) mechanisms to avoid threats and possible contradictory effects. This formulation is solver-independent. This means that any off-the-shelf CSP solver that supports the expressiveness of our formulation, with binary and non-binary constraints, can be used.

4.1 Variables

For each action a in $A?$, we create the seven kinds of variables specified in Table 1. Variables define the timestamps for actions, the causal links, the interval when conditions must hold and the time when the effects happen. For simplicity, and to deal with integer variables, we model time in $\mathbb{Z}^+$. To prevent time from exceeding the plan horizon, we bound all times to the makespan of the plan.[7]

6. Static information is commonly used in planning for the grounding process, e.g. to model that there is a link between two locations and, consequently, a driving is possible; to represent that level1 is before level2; to model a petrol station that allows a refuel action in a given location; etc.

7. We use the makespan, which can be observed, to restrict the duration of the actions. However, it is dispensable if we consider a long enough domain for durations.

TABLE 1
Formulation of Variables and Their Domains for Actions in A?

Variable	Domain	Description
$\mathsf{start}(a)$	$[0..makespan]$	start time of a that might be observed in O or derived
$\mathsf{dur}(a)$	$[0..makespan]$	duration of a
$\mathsf{end}(a)$	$[0..makespan]$	end time of a that might be observed in O or derived
$\mathsf{sup}(p,a)$	$\{b_i\}$ that supports p	symbolic variable for the set of potential supporters b_i of condition p of a (causal link $\langle b_i, p, a\rangle$)
$\mathsf{req\_start}(p,a)$, $\mathsf{req\_end}(p,a)$	$[0..makespan]$	interval $[\mathsf{req\_start}(p,a)..\mathsf{req\_end}(p,a)]$ at which action a requires p
$\mathsf{time}(p,a)$	$[0..makespan]$	time when effect p of a happens

Our temporal model formulation is more expressive than PDDL2.1 (see more details in Section 4.3), and allows conditions and effects to be at any time, even outside the execution of the action. For instance, let us imagine a condition p that only needs to be maintained for 5 time units before an action a starts (e.g. warming-up a motor before driving): the expression $\mathsf{req\_end}(p,a) = \mathsf{start}(a); \mathsf{req\_end}(p,a) = \mathsf{req\_start}(p,a) + 5$ is possible in our formulation. Additionally, we can represent an effect p that happens in the middle of action a: $\mathsf{time}(p,a) = \mathsf{start}(a) + (\mathsf{dur}(a)/2)$ is also possible.

Additionally, we create two dummy actions init and goal for each planning problem $\langle F, I, G, A\rangle$. First, init represents the initial state I ($\mathsf{start}(\mathsf{init}) = 0$ and $\mathsf{dur}(\mathsf{init}) = 0$). init has no variables $\mathsf{sup}, \mathsf{req\_start}$ and $\mathsf{req\_end}$ because it has no conditions. init has as many $\mathsf{time}(p_i, \mathsf{init}) = 0$ as p_i in I. Second, goal represents G ($\mathsf{start}(\mathsf{goal}) = makespan$ and $\mathsf{dur}(\mathsf{goal}) = 0$). goal has as many $\mathsf{sup}(p_i, \mathsf{goal})$ and $\mathsf{req\_start}(p_i, \mathsf{goal}) = \mathsf{req\_end}(p_i, \mathsf{goal}) = makespan$ as p_i in G. goal has no variables time as it has no effects.

This formulation allows us to model TILs exactly like other actions. $\mathsf{til}(f,t)$ can be seen as a dummy action ($\mathsf{start}(\mathsf{til}(f,t)) = t$ and $\mathsf{dur}(\mathsf{til}(f,t)) = 0$) with no conditions and only one effect f that happens at time t ($\mathsf{time}(f, \mathsf{til}(f,t)) = t$). A til is similar to init, as they both represent information that is given at a particular time, but externally to the execution of the plan.

4.2 Constraints

Table 2 shows the constraints that we define among the variables of Table 1. The three first constraints are intuitive enough. The fourth constraint models the causal links. Note that in a causal link $\langle b_i, p, a\rangle$, $\mathsf{time}(p, b_i) < \mathsf{req\_start}(p,a)$ and not $\leq$. This is because temporal planning assumes an $\epsilon > 0$, as a small tolerance that implies no collision between the time when an effect p is supported and when it is required [3]. When time is modeled in $\mathbb{R}^+$, epsilon is usually 0.001 but when it is modeled in $\mathbb{Z}^+$, $\epsilon = 1$ and $\leq$ becomes $<$. The fifth constraint avoids any threat via promotion or demotion [2]. The sixth constraint models the fact the same action requires and deletes p. Note the $\geq$ inequality here; this is possible because if one condition and one effect of the same action a happen at the same time, the underlying semantics in planning considers the condition in a is checked instantly before the effect in a [3]. The seventh constraint solves the fact that two (possibly equal) actions have contradictory effects.

It is important to note that the constraints involve any type of actions. Consequently, init, goal and til are subsumed in this formulation.

4.3 Specific Constraints for Durative Actions of PDDL2.1

As Section 2.2 explains, PDDL2.1 restricts the expressiveness of temporal planning in terms of conditions, effects, durations and structure of the actions. Hence, our temporal formulation subsumes and is significantly richer than PDDL2.1; but adding constraints to make it fully PDDL2.1-compliant is straightforward.

First, adding $((\mathsf{req\_start}(p,a) = \mathsf{start}(a))$ OR $(\mathsf{req\_start}(p,a) = \mathsf{end}(a)))$ AND $((\mathsf{req\_end}(p,a) = \mathsf{start}(a))$ OR $(\mathsf{req\_end}(p,a) = \mathsf{end}(a)))$ limits condition p to be *at start*, *over all* or *at end*, i.e. p is in $\mathsf{cond}_s(a)$, $\mathsf{cond}_o(a)$ or $\mathsf{cond}_e(a)$, respectively. Further, if a condition p is never deleted in a plan, it can be considered as static or invariant information, and the constraint to be added is simply: $((\mathsf{req\_start}(p,a) = \mathsf{start}(a))$ AND $(\mathsf{req\_end}(p,a) = \mathsf{end}(a)))$, i.e. $p \in \mathsf{cond}_o(a)$. Surprisingly, invariant conditions are modeled differently depending on the human modeler. See, for instance, `(link ?from ?to)` of Fig. 2, which is modeled as an *at start* condition despite: i) the link should be necessary all over the driving; and ii) no action in this domain deletes that link. This also happens in the *transport* domain of the IPC, where a `refuel` action requires to have a petrol station in a location only *at start*, rather than *over all* which makes more sense. This shows that modeling the temporal features is not easy even for an expert and it highly depends on the human's decision. On the contrary, our formulation checks the invariant conditions and deals with them always in the same coherent way.

Second, $((\mathsf{time}(p,a) = \mathsf{start}(a))$ OR $(\mathsf{time}(p,a) = \mathsf{end}(a)))$ makes an effect p happen only *at start* or *at end* of action a, i.e. p is in $\mathsf{eff}_s(a)$ or $\mathsf{eff}_e(a)$. Also, if all effects happen *at start* the duration of the action would be irrelevant and could exceed the plan makespan. To avoid this, for any action a, at least one of its effects should happen *at end*: $\sum_{i=1}^{n=|\mathsf{eff}(a)|} \mathsf{time}(p_i, a) > n \times \mathsf{start}(a)$, which guarantees $\mathsf{eff}_e(a)$ is not empty.

Third, durations in PDDL2.1 can be defined in two different ways. On the one hand, durations can be equal for all grounded actions of the same operator. For instance, any instantiation of `board-truck` of Fig. 2 will last 2 time units no matter its parameters. Although this may

TABLE 2
Formulation of Constraints.

Constraint	Description
$\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$	relation between start and end time of a
$\mathsf{end}(a) \leq \mathsf{start}(\mathsf{goal})$	goal is always the last action of the plan
$\mathsf{req\_start}(p,a) \leq \mathsf{req\_end}(p,a)$	interval [$\mathsf{req\_start}(p,a)$..$\mathsf{req\_end}(p,a)$] must be a valid interval
if $\mathsf{sup}(p,a) = b_i$ then $\mathsf{time}(p,b_i) < \mathsf{req\_start}(p,a)$	modeling causal link $\langle b_i, p, a\rangle$: the time when b_i supports p must be before a requires p
$\forall b_j \neq a$ that deletes p at time τ_j: if $\mathsf{sup}(p,a) = b_i$ then $\tau_j < \mathsf{time}(p,b_i)$ OR $\tau_j > \mathsf{req\_end}(p,a)$	solving threat of b_j to causal link $\langle b_i, p, a\rangle$ being $b_j \neq a$ (promotion OR demotion)
if a requires and deletes p: $\mathsf{time}(not-p,a) \geq \mathsf{req\_end}(p,a)$	when a requires and deletes p, the effect cannot happen before the condition
$\forall a_i, a_j \mid a_i$ supports p and a_j deletes p: $\mathsf{time}(p,a_i) \neq \mathsf{time}(not-p,a_j)$	solving effect interference (p and $not-p$): they cannot happen at the same time

seem a bit odd, it is not an uncommon practice to simplify the model. The constraint to model this is: $\forall a_i, a_j$ being instances of the same operator: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$. On the other hand, although different instantiations of `drive-truck` will last different depending on the locations, different occurrences of the same instantiated action will last equal. In a PDDL2.1 temporal plan, multiple occurrences of `drive-truck(truck1,loc1,loc2,driver1)` will have the same duration no matter when they start. Intuitively, they are different occurrences of the same action, but in the real-world the durations would differ from driving at night or in peak times. Since PDDL2.1 makes no distinction among different occurrences, the constraint to add is: $\forall a_i, a_j$ being occurrences of the same durative action: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$. Obviously, this second constraint is subsumed by the first one in the general case where all instances of the same operator have the same duration.

Fourth, the structure of conditions and effects for all grounded actions of the same operator is constant in PDDL2.1. This means that if `(empty ?t)` is an *at start* condition of `board-truck`, it will be *at start* in any of its grounded actions. Let $\{p_i\}$ be the conditions of an operator and $\{a_j\}$ be the instances of a particular operator. The following constraints are necessary to guarantee a constant structure:

$\forall p_i : (\forall a_j : \mathsf{req\_start}(p_i,a_j) = \mathsf{start}(a_j))$ OR $(\forall a_j : \mathsf{req\_start}(p_i,a_j) = \mathsf{end}(a_j))$

$\forall p_i : (\forall a_j : \mathsf{req\_end}(p_i,a_j) = \mathsf{start}(a_j))$ OR $(\forall a_j : \mathsf{req\_end}(p_i,a_j) = \mathsf{end}(a_j))$

And analogously for all effects $\{p_i\}$ and the instances $\{a_j\}$ of an operator:

$\forall p_i : (\forall a_j : \mathsf{time}(p_i,a_j) = \mathsf{start}(a_j))$ OR $(\forall a_j : \mathsf{time}(p_i,a_j) = \mathsf{end}(a_j))$

As a conclusion, in our formulation each action of $A?$ is modeled separately so it does not need to share the same structure or duration of other actions. Moreover, the time-stamps for conditions/effects can be arbitrarily placed inside or outside the execution of the action, which allows for a flexible and expressive temporal model. But, when necessary, we can simply include additional constraints to restrict the expressiveness of the model, such as the ones provided by PDDL2.1, because the operator is directly extracted from the action's name.

4.4 Example

We now show a fragment of the formulation for the example depicted in Fig. 3. For simplicity, we only show the variables and constraints for action $a3$, but the formulation is analogous for all other actions.

The variables and domains are: $\mathsf{start}(a3) = 7$; $\mathsf{dur}(a3) \in [1..15]$; $\mathsf{end}(a3) = \mathsf{start}(a3) + \mathsf{dur}(a3)$; $\mathsf{sup}(r,a3) \in \{a1\}$; $\mathsf{req\_start}(r,a3), \mathsf{req\_end}(r,a3) \in [0..22]$; and $\mathsf{time}(not-p,a3) \in [0..22]$. On the other hand, the constraints are: $\mathsf{end}(a3) \leq \mathsf{start}(\mathsf{goal})$; $\mathsf{req\_start}(r,a3) \leq \mathsf{req\_end}(r,a3)$; if $\mathsf{sup}(r,a3) = a1$ then $\mathsf{time}(r,a1) < \mathsf{req\_start}(r,a3)$; if $\mathsf{sup}(r,a3) = a1$ then $((\mathsf{time}(not-r,a4) < \mathsf{time}(r,a1))$ OR $(\mathsf{time}(not-r,a4) > \mathsf{req\_end}(r,a3)))$; $\mathsf{time}(not-p,a3) \neq \mathsf{time}(p,a1)$ and $\mathsf{time}(not-p,a3) \neq \mathsf{time}(p,a2)$.

There is an unaffordable number of solutions to this simple example, mainly because there is a huge range of possible durations that make the learned model consistent with the partially specified model $A?$. A solution has assigned consistent values to all variables of Table 1 and satisfies all the constraints of Table 2 for this example. Fig. 4 shows six solution models, randomly chosen from the first 100 solutions. What is important to note is that the structure, i.e. distribution of conditions/effects, is similar in all the learned models. Actually, the distribution of the effects is identical (except for q in model 2), and the distribution of conditions is very similar (e.g. q is always in cond_o and r in $a4$ is very often in cond_o). Obviously, learned models tend to be similar, which is very positive. Indiscriminately changing a single feature means a different, though similar, learned model that might fail to satisfy the constraints and lead to an inconsistency. The durations are, however, more different: $\mathsf{dur}(a1)$ ranges in these models from 7 to 19, whereas $\mathsf{dur}(a2)$ ranges from 5 to 18. As explained in Section 3.2, learning the precise duration from just one sample may not be always possible, which is the main limitation of the one-shot learning task. In fact, the specific constraint of PDDL2.1, with regard to having multiple occurrences of the same action having the same duration, can significantly

help us to learn the actions' duration in a more precise way, because the learned duration must be consistent with all those occurrences.

4.5 Implementation. Use of Heuristics for Resolution

Our CSP formulation is automatically compiled from a partially specified action model, as defined in a classical planning problem, and the observations from a plan execution. The formulation has been implemented in Choco[8], an open-source Java library for CP that provides an object-oriented API to state the constraints to be satisfied.

Our formulation is solver-independent, which means we do not use heuristics that may require changes in the implementation of the CSP engine. Although this reduces the solver's performance, we are mainly interested in using it as a blackbox that can be easily changed with no modification in our formulation. However, we can easily encode standard static heuristics for variable and value selection that help improve efficiency by following the next ordering, which has shown very efficient in our experiments:

1) Effects (time). For negative effects, first the lower value and for positive effects, first the upper value. This gives priority to delete effects as $\mathsf{eff}_s(a)$ and positive effects as $\mathsf{eff}_e(a)$.
2) Conditions (req_start and req_end). For req_start, first the lower value, whereas for req_end, first the upper value. This gives priority to $\mathsf{cond}_o(a)$, trying to keep the conditions as long as possible.
3) Supporters (sup). First the lower value, thus preferring the supporter that starts earlier in the plan.
4) Duration (dur). First the lower value, thus applying the principle of the shortest actions that make the learned model consistent.

4.6 Using the CP Formulation for Plan Validation

We explained that adding extra constraints allows us to restrict the temporal expressiveness of the learned model. We show here that we can also restrict the learned model by constraining the variables to known values, which is specially interesting when there is additional information on the temporal model that needs to be represented. For instance, based on past learned models, we may know the precise duration of an action a is 6, or we can figure out that its effect p always happens at end. Our CP formulation can include this by simply adding $\mathsf{dur}(a) = 6$ and $\mathsf{time}(p, a) = \mathsf{end}(a)$, respectively, which is useful to enrich the partially specified actions in $A?$ of the learning task.

In particular, the possibility of adding those constraints is very appealing when used for validating whether a partial action model allows us to learn a consistent model, as we will see in Section 5. This leads us to a unified formulation for learning and validation. Let us assume that the distribution of all (or just a few) conditions and/or effects is known and, in consequence, represented in the learning task. If a solution is found, then that structure of conditions/effects is consistent for the learned model. On the contrary, if no solution is found that structure is inconsistent and cannot be explained. Analogously, we can represent known values for the durations. If a solution is found, the durations are consistent, and inconsistent otherwise. Hence, we have three options for validating a partial model *w.r.t.*: i) a known structure with the distribution of conditions/effects; ii) a known set of durations; and iii) a known structure plus a known set of durations (i+ii). The first and second option allows for some flexibility in the learning task because some variables remain open. On the contrary, the third option checks whether a learned model can fit the given constraints, thus reproducing a plan validation task equivalent to [5], but now much more expressive.

8. Choco is available at `www.choco-solver.org`

5 Evaluation

This section evaluates our approach from two points of view. First, we assess the performance of the CP formulation for solving the learning task. Second, we assess the quality of the resulting learned model.

5.1 Setup

We have run 569 different instances, from the simplest to the most complex ones, on nine IPC planning domains. These domains are encoded in PDDL2.1, so we have included the constraints given in Section 4.3. We first get the plans for these domains by using five planners (*LPG-Quality* [25], *LPG-Speed* [25], *TP* [26], *TFD* [27] and *TFLAP* [28]), where the planning time is limited to 100s. The actions and their start time observations on each plan are automatically compiled into a CSP learning instance. Then, we run the one-shot learning task to get the action model (with the temporal features) for each instance, where the learning time is limited to 100s on an Intel i5-6400 @ 2.70GHz with 8GB of RAM. Since each CSP instance can have many solutions, we always select the first solution found.

5.2 Performance Evaluation

We measure the performance of the learning task in terms of the size of the CP formulation, according to the variables and constraints defined in Section 4. Table 3 shows the number of instances tested per domain, and the min-max (and average between brackets) number of variables and constraints of the formulation for all the instances. The last column shows the average resolution time per instance. For example, the *zenotravel* domain contains 78 instances, which means compiling 78 formulations. The formulations' size ranges from 98 to 2878 variables and from 352 to 56402 constraints. The average resolution time per formulation is 0.02s.

In all the domains, the number of variables is high, but since they are highly constrained the resolution time remains very low. This is a consequence of the one-shot learning approach, which only needs to include the variables+constraints of one plan instance, thus requiring less computation time.

5.3 Quality Evaluation

The quality evaluation of the learned model can be addressed from two perspectives. From a pure syntactic perspective, learning can be considered as an automated design

Action	dur	cond_s	cond_o	cond_e	eff_s	eff_e
Learned model 1						
$a1$	8				r	p
$a2$	18				q	p
$a3$	1		r			$not-p$
$a4$	1		q,r	p		$not-r$
Learned model 2						
$a1$	19				r	p
$a2$	5					p,q
$a3$	1			r		$not-p$
$a4$	1	r	p,q			$not-r$
Learned model 3						
$a1$	7				r	p
$a2$	18				q	p
$a3$	1			r		$not-p$
$a4$	1		p,q,r			$not-r$

Action	dur	cond_s	cond_o	cond_e	eff_s	eff_e
Learned model 4						
$a1$	7				r	p
$a2$	9				q	p
$a3$	1			r		$not-p$
$a4$	1		p,q,r			$not-r$
Learned model 5						
$a1$	9				r	p
$a2$	6				q	p
$a3$	1		r			$not-p$
$a4$	1		q,r	p		$not-r$
Learned model 6						
$a1$	8				r	p
$a2$	16				q	p
$a3$	1	r				$not-p$
$a4$	1		q,r	p		$not-r$

Fig. 4. Six learned models, randomly chosen from the first 100 solutions, to the example of Fig. 3.

TABLE 3
CP Formulation Details for the Instances of IPC Domains.

Domain	#instances	#variables (average)	#constraints (average)	average resolution time (s)
zenotravel	78	98-2878 (825)	352-56402 (8520)	0.02
driverlog	73	193-14043 (2862)	847-170146 (18457)	0.08
depots	64	242-2943 (1142)	1577-430688 (20318)	0.04
rovers	84	290-9307 (1991)	1010-48501 (10540)	0.20
satellite	84	174-3054 (1189)	967-19709 (6294)	0.05
storage	69	99-3479 (704)	3023-597178 (17213)	0.03
floortile	17	2564-7994 (4647)	14073-847609 (129881)	0.20
parking	49	539-1275 (834)	2581-7366 (4208)	0.03
sokoban	51	8586-57779 (21142)	35882-390316 (97133)	0.44

task to create a new model that is similar to a reference (or *ground truth*) model. The aim is to assess the precision or accuracy of the learned model, a common metric in learning [7], [21], [24]. From a semantic perspective, learning can be considered as a classification task, where we first learn a model from a training dataset and then validate it on a test dataset. The aim is to assess how much the learned model explains (or transfers) to unseen test samples [10], [15], [29].

5.3.1 Syntactic Evaluation. Precision

Precision in learning is a similarity measure between two models, one reference and one learned model. In our case, $precision = 100 \times \frac{f^{=}}{f^{=}+f^{\neq}}$, where $f^{=}$ counts the number of facts (i.e. conditions+effects) that are temporally distributed equally in both models, and $f^{\neq}$ counts the number of facts that are distributed in a different way. As an example, given an action or operator, if p is an *at start* condition in the reference model and also in the learned model, it counts as a hit in $f^{=}$, and in $f^{\neq}$ if they differ. This is repeated for all conditions and effects. A precision of 100% means the `:condition` and `:effect` sections in both models are, respectively, syntactically identical. Roughly speaking, the learned structure of conditions/effects matches exactly the reference model.

Table 4 shows the syntactic evaluation for our 569 learned models, including the number of operators (or action schemas), the total number of facts ($f^{=} + f^{\neq}$) to distribute within the operators, and the average precision scores. We first use as the reference model the hand-written domain as provided in IPC. On average, the precision scores

TABLE 4
Precision *w.r.t.* a REFerence and a More Rational REVised Model.

Domain	#ops	#facts	REF model	REV model
zenotravel	5	28	61.04%	83.99%
driverlog	6	28	76.52%	83.71%
depots	5	37	74.66%	75.25%
rovers	9	69	49.63%	84.64%
satellite	5	24	71.72%	71.72%
storage	5	38	80.99%	81.37%
floortile	7	44	81.42%	81.42%
parking	4	31	77.46%	77.46%
sokoban	3	33	97.80%	97.80%

are very good, taking into consideration that the learning is done under a one-shot approach. The ratios range from 49.63% in *rovers*, the domain with the highest number of operators and facts, to 97.80% for *sokoban*, the domain with the lowest number of operators.

Unfortunately, there is not a unique reference model when learning real-world temporal models; e.g. `full` and `not-empty` effects can be interchangeable in some domains, as they represent the same information, but they are syntactically different. Also, a pure syntax-based measure may return misleading results, as it may count as incorrect ($f^{\neq}$) a change in the distribution of conditions/effects that represents an equivalent reformulation of the reference model. For instance, given the example of Fig. 2, the condition learned `(over all (link ?from ?to))` would be counted as a difference for action `drive-truck`, as it is *at start* in the reference model; but it is, rationally speaking,

even more correct. This misleading situation is common in IPC, where the domains' definition is not always rational due to: i) *at start* conditions that should be *over all* (in *zenotravel*, *driverlog* and *rovers*); ii) *at start* effects that should be *at end*, and vice versa (in *zenotravel*, *depots*, *rovers* and *satellite*); and iii) actions with only *at start* effects that make durative actions unnecessarily tricky (in *depots*). As a knowledge engineering step, we have revised all the domains to make them more rational and recalculated the precision. The results are shown in the last column of Table 4. Learned models are significantly more precise in the revised domains (specially in *zenotravel* and *rovers*), ranging now from 71.72% to 97.80%. This means the `:condition` and `:effect` sections are, respectively, syntactically identical in more than 70% of the revised *vs.* learned models. Also, 100% of the *over all* conditions that represent static information is precisely learned, thus being more coherent than human designers. In *satellite* however, the results remain the same, as the learned models still mislead some inner effects within the actions. Obviously, the reference domains that are well defined and need no revision (*floortile*, *parking* and *sokoban*) return the same precision scores.

5.3.2 Semantic Evaluation. Explanation and Validation

As pointed out above, the temporal ground truth about the temporal world is not unique. A precision score of 75% might still be an excellent result, provided the learned model explains all the observations of the plan trace; that is, the model is consistent because it satisfies all the constraints of the learning task. Also, as seen in Section 3.2, some learned durations cannot be granted and will differ from a reference model, but the underlying model is still consistent. For instance, the six learned models of Fig. 4 are not syntactically identical, but they are all consistent for the example of Fig. 3. Therefore, a syntactic evaluation in learning is a bit limited and we should perform a further semantic evaluation. From this standpoint, the quality of the learned model can be assessed by analyzing the success ratio of the learned model *vs.* unseen samples of a test dataset, analogously to a classification task. We define the success ratio as $success = 100 \times \frac{samples^{\checkmark}}{|dataset|}$, where $samples^{\checkmark}$ counts the number of samples the learned model explains on a test $dataset$. A success of 100% implies learning a model that explains the full dataset: a feasible solution is found which is consistent with the constraints of the learned model together with the test samples ones. But if, for example, one condition is learned as *at start* but it leads to an inconsistency in a test sample (where it must remain *over all*), this does not count in $samples^{\checkmark}$.

Given the learned model for an instance of a domain, we assess the quality of such a model *vs.* a dataset formed by the remaining instances of that domain. For example, the *zenotravel* domain contains 78 instances, as indicated in Table 3, which means learning 78 models. Each model is evaluated by using the 77 remaining models, thus producing 78×77=6006 tests. Each test is repeated three times to validate each model *vs.* the other models *w.r.t.* the *struct*ure, the *dur*ation and the *struct*ure+*dur*ation, as discussed in Section 4.6. Table 5 shows the average success ratio for our IPC domains.

TABLE 5
Average Success of the One-shot Learned Model *vs.* the Test Dataset.

Domain	#tests	struct	dur	struct+dur
zenotravel	6006	100%	68.83%	35.76%
driverlog	5256	97.60%	44.86%	21.04%
depots	4032	55.41%	76.22%	23.19%
rovers	6972	78.84%	5.35%	0.17%
satellite	6972	80.74%	57.13%	40.53%
storage	4692	58.08%	70.10%	38.36%
floortile	272	100%	80.88%	48.90%
parking	2352	86.69%	81.38%	54.89%
sokoban	2550	100%	87.25%	79.96%

In *zenotravel*, the struct value means that the distribution of conditions/effects learned by using only one plan sample is consistent with all the samples used as dataset (100% of the 6006 tests), which is the perfect result, as also happens in *floortile* and *sokoban* domains. The dur value means the durations learned explain 68.83% of the dataset. This value is usually lower because any learned duration that leads to inconsistency in a sample counts as a failure. The struct+dur value means that the learned model explains entirely 35.76% of the samples. This value is always the lowest because a subtle change in the structure or duration learned that leads to inconsistency counts as a failure. The results depend on the domain size (number of operators, which need to be grounded), the relationships (causal links, threats and interferences) among the actions, and the size and quality of the plans. The worst result is returned in the *rovers* domain, which models a group of planetary rovers to explore the planet they are on. Since there are many facts, parallel actions for taking pictures/samples and navigation of multiple rovers, learning the duration and the structure+duration is particularly complex in this domain.

In general, the quality of the learned models is good enough to explain a significant number of test samples, specially in terms of the structure. This is interesting, taking into consideration that the one-shot approach uses only one sample to learn the temporal features. However, learning the durations proves more complex. In some cases, we have observed that planners return plans with unnecessary actions, which has a negative impact for learning precise durations, as happens in *rovers*.

6 Conclusions

The interest in learning is growing up because it allows us to acquire procedural knowledge through demonstration and partial observations. Learning planning action models by observations of plan traces is useful in many scenarios: recognition of past behavior for prediction and anticipation, decision taking and recommendation, programming and modeling, teleoperation, macro recording, sensing and controlling, robotics motion capturing and planning, etc. Learning is appealing because these scenarios include a huge number of tasks, sometimes difficult to be described formally (and more difficult to be annotated temporally), which require expert knowledge and engineering that becomes impractical in complex domains.

In this paper we have presented a purely declarative CP formulation, independent of any CSP solver, to address the automated learning of temporal features on action models which, to our knowledge, initiates a novel approach for the intersection of knowledge engineering, learning, CP and planning. Metaphorically speaking, learning the classical model is to planning what learning the temporal features is to temporal planning. Knowing these features is useful in practice, where learning a classical planning model is not enough and needs to be extended with temporal features to increase its applicability. In consequence, learning the temporal features bridges the gap between the planning action model and the real (temporal) world.

Our main result is an effective formulation that is automatically derived, without the necessity of specific hand-coded domain knowledge. The ultimate goal of this learning is to reduce the laborious modeling stage, to minimize the effort human experts need to design temporal planning models before launching the planners. Our formulation is flexible enough to accommodate different types of observations as input information/knowledge, and supports a rich temporal planning model with concurrency; although its expressiveness is beyond PDDL2.1, it can be easily modified to be PDDL2.1-compliant. Formal properties are inherited from the formulation itself and the CSP solver. The formulation is correct because the definition of constraints to solve causal links, threats and effect interferences are supported, which avoids contradictions. It is also complete because the solution needs to be consistent with all the imposed constraints, while a complete exploration of the domain of each variable returns all the possible learned models in the form of alternative consistent solutions.

Unlike other approaches that need to learn from datasets with many samples, we perform a one-shot learning task. Large datasets could lead to better learned models but: i) retrieving many samples is not always easy, specially in human interactive environments that require learning by demonstration; and ii) this would need to estimate the best number of samples and to learn a model that explains the highest number of samples, involving an optimization process, more expensive than satisfaction. Our approach reduces both the size of the required datasets and the computation time. Note, however, that we can easily deal with a large number of input samples by simply including more input observations in our learning task, i.e. by using very long plan traces. And this requires no changes in our CP formulation.

According to our syntactic+semantic evaluation, the one-shot learned models are precise and explain a high number of samples in the datasets used for testing. Moreover, the same CP formulation is valid for learning and for validation, by simply adding constraints to the variables. This is an innovative advantage, as the same formulation allows us to carry out different tasks: from entirely learning, partial learning/validation (structure and/or duration) to entirely plan validation. From our experiments, learning the structure of the actions in a one-shot way leads to representative enough models, but learning the precise durations is more difficult when irrelevant actions are planned.

Finally, it is important to note that our CP formulation can be represented and solved by Satisfiability Modulo Theories, which is part of our current work. As for future work, we want to extend our formulation to learn from intermediate observations (we need to investigate how many and how frequent they must be), to learn meta-models (as combinations of several learned models), and to learn more complete action models. In the latter, we will relax the input action model to find out the conditions/effects together with their temporal distribution. This implies to remove constraints from the partially specified set of actions in $A?$ and to investigate the dependencies between the conditions/effects and temporal features (e.g. some effects could conditionally happen depending on temporal constraints). The underlying idea of finding an action model consistent with all the constraints will remain exactly the same, but the model will need to be extended with additional decision variables ($\mathsf{is\_condition}(p, a)$ and $\mathsf{is\_effect}(p, a)$) and constraints to decide whether p is a condition or effect of action a. This will provide a very open model of actions (we will need to investigate whether the learned model is still precise and valid) that will lead to the analysis of new heuristics for its resolution.

REFERENCES

[1] H. Geffner and B. Bonet, "A concise introduction to models and methods for automated planning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 1, pp. 1–141, 2013.

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

[3] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[4] W. Cushing, S. Kambhampati, D. S. Weld *et al.*, "When is temporal planning really temporal?" in *Proc. of the International Joint Conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 2007, pp. 1852–1859.

[5] R. Howey, D. Long, and M. Fox, "VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL," in *Proc. of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-04)*, 2004, pp. 294–301.

[6] S. Kambhampati, "Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models," in *Proc. of the National Conference on Artificial Intelligence (AAAI-07)*, vol. 22(2), 2007, pp. 1601–1604.

[7] H. H. Zhuo, Q. Yang, D. H. Hu, and L. Li, "Learning complex action models with quantifiers and logical implications," *Artificial Intelligence*, vol. 174, no. 18, pp. 1540–1569, 2010.

[8] C. L. Baker, R. Saxe, and J. B. Tenenbaum, "Action understanding as inverse planning," *Cognition*, vol. 113, pp. 329–349, 2009.

[9] S. Jiménez, T. De la Rosa, S. Fernández, F. Fernández, and D. Borrajo, "A review of machine learning for automated planning," *The Knowledge Engineering Review*, vol. 27, no. 4, pp. 433–467, 2012.

[10] M. Aggarwal, "On learning of choice models with interactive attributes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28(10), pp. 2697–2708, 2016.

[11] J. Kucera and R. Barták, "LOUGA: learning planning operators using genetic algorithms," in *Proc. of the Pacific Rim Knowledge Acquisition Workshop (PKAW-18)*, 2018, pp. 124–138.

[12] K. Mourão, L. S. Zettlemoyer, R. P. A. Petrick, and M. Steedman, "Learning STRIPS operators from noisy and incomplete observations," in *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI-12)*, 2012, pp. 614–623.

[13] Q. Yang, K. Wu, and Y. Jiang, "Learning action models from plan examples using weighted MAX-SAT," *Artificial Intelligence*, vol. 171, no. 2-3, pp. 107–143, 2007.

[14] H. H. Zhuo and S. Kambhampati, "Action-model acquisition from noisy plan traces," in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-13)*, 2013, pp. 2444–2450.

[15] S. Jialin Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22(10), pp. 1345–1359, 2010.

[16] A. Arora, H. Fiorino, D. Pellier, M. Métivier, and S. Pesty, "A review of learning planning action models," *The Knowledge Engineering Review*, vol. 33, 2018.
[17] E. Amir and A. Chang, "Learning partially observable deterministic action models," *Journal of Artificial Intelligence Research*, vol. 33, pp. 349–402, 2008.
[18] S. N. Cresswell, T. L. McCluskey, and M. M. West, "Acquiring planning domain models using LOCM," *The Knowledge Engineering Review*, vol. 28, no. 02, pp. 195–213, 2013.
[19] H. H. Zhuo and S. Kambhampati, "Model-lite planning: Case-based vs. model-based approaches," *Artificial Intelligence*, vol. 246, pp. 1–21, 2017.
[20] H. H. Zhuo, T. A. Nguyen, and S. Kambhampati, "Refining incomplete planning domain models through plan traces," in *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-13)*, 2013, pp. 2451–2458.
[21] H. H. Zhuo, H. Muñoz Avila, and Q. Yang, "Learning hierarchical task network domains from partially observed plan traces," *Artificial Intelligence*, vol. 212, pp. 134–157, 2014.
[22] A. Garrido, M. Arangu, and E. Onaindia, "A constraint programming formulation for planning: from plan scheduling to plan generation," *Journal of Scheduling*, vol. 12, no. 3, pp. 227–256, 2009.
[23] J. Hoffmann and S. Edelkamp, "The deterministic part of IPC-4: an overview," *Journal of Artificial Intelligence Research*, vol. 24, pp. 519–579, 2005.
[24] D. Aineto, S. Jiménez, and E. Onaindia, "Learning STRIPS action models with classical planning," in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS-18)*, 2018, pp. 399–407.
[25] A. Gerevini, A. Saetti, and I. Serina, "Planning through stochastic local search and temporal action graphs in lpg," *Journal of Artificial Intelligence Research*, vol. 20, pp. 239–290, 2003.
[26] S. Jiménez, A. Jonsson, and H. Palacios, "Temporal planning with required concurrency using classical planning," in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS-15)*, 2015.
[27] P. Eyerich, R. Mattmüller, and G. Röger, "Using the context-enhanced additive heuristic for temporal and numeric planning," in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS-09)*, 2009.
[28] E. Marzal, L. Sebastia, and E. Onaindia, "Temporal landmark graphs for solving overconstrained planning problems," *Knowledge-Based Systems*, vol. 106, pp. 14–25, 2016.
[29] H. H. Zhuo and Q. Yang, "Action-model acquisition for planning via transfer learning," *Artificial Intelligence*, vol. 212, pp. 80–103, 2014.

Antonio Garrido received his degree and European PhD in Computer Science from the Universitat Politecnica de Valencia, Spain, in 1998 and 2003, respectively. He is an associate professor in the same university. His research areas of interest include techniques of AI, mainly those related to search such as planning and, particularly, temporal planning, scheduling, constraint satisfaction, heuristics and their multiple applications to learning, e-learning, optimization and recommendation. He has published extensively in his areas of interest in journals and conferences. More details about his research and background can be found at https://agarridot.blogs.upv.es.

Sergio Jiménez received the MS and BS degrees in 2002 and 2004, respectively, from the Universidad Autónoma de Madrid, Spain. He received his European PhD in Computer Science, which obtained the distinguished Thesis Award, in 2011 from the Universidad Carlos III de Madrid, Spain. He was a post-doc at the University of Melbourne, Australia, and a Juan de la Cierva fellow at the Universitat Pompeu Fabra de Barcelona, Spain. Since 2017 he is a Ramón y Cajal fellow at the Universitat Politecnica de Valencia, Spain. His research areas of interest include finding innovative integrations of learning, machine learning and automated planning to improve the strengths of AI paradigms. He has published extensively in his areas of interest in journals and conferences. More details about his research and background can be found at http://serjice.webs.upv.es.