# One-Shot Learning of Concurrent Action Models

**Antonio Garrido** and **Sergio Jiménez**

**Abstract.** We present a *constraint programming* (CP) formulation for learning models of planning actions that can be executed in parallel and overlap. With the aim of understanding better the connection between the learning of planing action models, the synthesis of plans and the plan validation task, this paper studies a singular learning scenario where just a single observation of a plan execution (*one-shot*) is available. Our CP formulation is inspired by previous approaches for *temporal planning* so it models time-stamps for actions, *causal link* relationships, *threats* and effect *interferences*. Further, our CP formulation is flexible to accommodate a different range of expressiveness, subsuming the PDDL2.1 temporal semantics, and it is solver-independent (i.e. off-the-shelf CSP solvers can be used for resolution).

## 1 Introduction

*Temporal planning* is an expressive planning model that relaxes the assumption of instantaneous actions of *classical planning* [9]. Actions in temporal planning are called *durative*, because each action has an associated duration and hence, actions conditions/effects may hold/happen at different times [6]. This means that actions in the temporal planning model can be executed in parallel and overlap in several ways [4] and that valid solutions for temporal planning instances must indicate the precise time-stamp when actions start and end [13].

Despite the potential of state-of-the-art planners, its applicability to the real world is still somewhat limited because of the difficulty of specifying correct and complete planning models [15]. The more expressive the planning model is, the more evident becomes this *knowledge acquisition bottleneck* that jeopardizes the usability of AI planning technology. This has led to a growing interest in the planning community for the learning of action models [20, 18, 21, 16]. Most of these approaches are however purely inductive and require large datasets of observations, e.g. hundreds of plan observations to compute statistically significant models that minimize some error metric over the input observations. What is more, when model learning is considered an optimization task, it cannot guarantee that the learned action models do not fail to explain a given input observation (or that the states induced by executing the learned actions are incorrect).

This paper follows a radically different approach and analyzes the application of *Constraint Programming* (CP) for the *one-shot learning* of temporal action models, that is, for the singular scenario where action models are learned from a single observation of the execution of a plan. The contributions of this work are two-fold:

1. As far as we know, this is the first approach for learning action models for temporal planning. Plan observations can refer to the execution of paralel and overlapping actions which makes our approach appealing for learning action models in multi-agent environments [7]. Learning classical action models from sequential plans is a well-studied problem that has been previously addressed by a wide range of different approaches [2]. Since pioneering learning systems like ARMS [20], we have seen systems able to learn action models with quantifiers [1, 24], from noisy actions or states [18, 21], from null state information [3], or from incomplete domain models [22, 23]. While learning an action model for classical planning means computing the actions' conditions and effects that are consistent with the input observations, learning *temporal action models* requires also: i) identifying how conditions and effects are temporally distributed in the action execution, and ii) estimate the action duration.
2. Our CP formulation connects the *learning* of planning action models with the *synthesis* and the *validation* of plans since it allows that of-the-shelf CSP solvers can be used for any of these tasks. Further, we show that the plan validation capacity of our CP formulation is beyond the functionality of VAL (the standard plan validation tool [13]) since it can address *plan validation* of partial (or even an empty) action models and with partially observed plan traces (VAL requires both a full plan and a full action model for plan validation).

As a motivating example, let us assume an observation of the executions of actions in a *logistics scenario*. Learning the actions model will allow us: i) to better understand the insights of the logistics in terms of what is possible (or not) and why, the model must be 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, such as public transit for commuters, tourists or people in general in metropolitan areas —*a.k.a.* smart urban mobility.

## 2 Background

This section formalizes the *temporal planning* model that we follow in this work, the *hypothesis space* that confines the set of possible action models for the learning task we address and the *sampling space* that defines the kind of plan observations that we handle.

### 2.1 Temporal Planning

We assume that *states* are factored into a set $F$ of Boolean variables. A state $s$ is a time-stamped full assignment of values to the variables in $F$.

A *temporal planning problem* is a tuple $\langle F, I, G, A\rangle$ where the *initial state* $I$, is a fully observed state (i.e. $|I| = |F|$) that is time-stamped with $t = 0$; $G \subseteq F$ is a conjunction of *goal conditions* over the variables in $F$ that defines the set of goal states and $A$ represents the set of *durative actions*. Durative actions have an associated duration and may have conditions/effects at different times [19, 8].

In this work we assume that the *durative actions* in $A$ are grounded from *action schemes* (also called *operators*) to compactly represent temporal planning problems. PDDL2.1 is a popular language for representing *temporal planning* problems and it is the input language for the temporal track of the International Planning Competition [6, 11]. According to PDDL2.1 a *durative action* $a \in A$ is defined with the following elements:

1. $\mathsf{dur}(a)$, a positive value indicating the *duration* of the action.
2. $\mathsf{cond}_s(a), \mathsf{cond}_o(a), \mathsf{cond}_e(a) \subseteq F$ representing the action *conditions*. Unlike the *pre*conditions of classical actions, action conditions in PDDL2.1 must hold: before $a$ is executed (*at start*), during the entire execution of $a$ (*over all*) or when $a$ finishes (*at end*), respectively. In the simplest case, $\mathsf{cond}_s(a) \cup \mathsf{cond}_o(a) \cup \mathsf{cond}_e(a) = \mathsf{pre}(a)$.[1]
3. $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$ represent the action *effects*. In PDDL2.1 effects may happen *at start* or *at end* of $a$, respectively (and can still be either positive or negative). Again, in the simplest case $\mathsf{eff}_s(a) \cup \mathsf{eff}_e(a) = \mathsf{eff}(a)$.

PDDL2.1 somewhat restricts the expressiveness of the temporal planning model since the semantics of a PDDL2.1 *durative action* can be defined in terms of just 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 $s$ with time-stamp $\mathsf{start}(a)$, then $\mathsf{cond}_s(a)$ must hold in $s$. Ending action $a$ in state $s'$, with time-stamp $\mathsf{end}(a)$, means $\mathsf{cond}_e(a)$ must hold in $s'$. *Over all* conditions must hold at any state between $s$ and $s'$ or, in other words, throughout interval $[\mathsf{start}(a)..\mathsf{end}(a)]$. Likewise, *at start* and *at end* effects are instantaneously applied at states $s$ and $s'$, respectively —continuous effects are not considered in this work.

A *temporal plan* is a set of pairs $\{(a_1, t_1), (a_2, t_2) \ldots (a_n, t_n)\}$. Each $(a_i, t_i)$ pair contains a durative action $a_i$ and a *time-stamp* $t_i = \mathsf{start}(a_i)$. The *execution* of a temporal plan starting from a given initial state $I$ induces a state sequence formed by the union of all states $\{s_{t_i}, s_{t_i+\mathsf{dur}(a_i)}\}$, where there exists an initial state $s_0 = I$, and a state $s_{end}$ that is the last state induced by the execution of the plan (sequential plans can then be expressed as temporal plans but not the opposite). We say that a temporal plan is a *solution* to a given temporal planning problem when its execution, starting from the corresponding initial state, eventually reaches a state that meets the goal conditions, $G \subseteq s_{end}$.

## 2.2 The hypothesis space

The target of the learning task addressed in this paper is a set of PDDL2.1 *durative* actions schemes. Fig. 1 shows an example of two schemes for PDDL2.1 *durative actions* taken from the *driverlog* domain. The schema `board-truck` has a fixed duration while the duration of `drive-truck` depends on the driving time associated to the two given locations.

Like in PDDL, we assume that the set $F$ of Boolean state variables is given by the instantiation of a given set of predicates $\Psi$. We denote as $\mathcal{I}_{\xi,\Psi}$ the *vocabulary* (set of symbols) that can appear in the *conditions* and *effects* of a given *durative* action schema $\xi$. This set is formally defined as the FOL interpretations of predicates $\Psi$, over the action parameters $pars(\xi)$.

```
(: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 ?l1 - location ?l2 - location ?d - driver)
  :duration (= ?duration (driving-time ?l1 ?l2))
  :condition (and (at start (at ?t ?l1)) (at start (link ?l1 ?l2))
                  (over all (driving ?d ?t)))
  :effect (and (at start (not (at ?t ?l1)))
               (at end (at ?t ?l2))))
```

**Figure 1.** Two action schemes for PDDL2.1 durative actions.

For a *durative* action schema $\xi$, the size of its space of possible action models is then $D \times 2^{5\times|\mathcal{I}_{\xi,\Psi}|}$ where D is the number of different possible durations for any action shaped by the $\xi$ schema. Note that this space is significantly larger than for learning STRIPS actions [20] where this number is $2^{2\times|\mathcal{I}_{\xi,\Psi}|}$ because negative effects must also be preconditions of the same action and cannot be positive effects of that action.

With this vocabulary defined then the *conditions* and *effects* of a *durative* action schema can be coded by 5 bit-vectors, each of length $|\mathcal{I}_{\xi,\Psi}|$. A 0-bit in the vector represents that the correpsonding condition of effect is not part of the schema while a 1-bit represents that is part of the schema. This also means that the *Hamming distance* can be used straightforward as a similarity metric for *durative* schemes. For instance, to compare a learned action model with respect to a given reference model that serves as baseline. The number of wrong 1-bits in the learned schema provide us a meassure of the incorrectness of the learned model and the number of wrong 0-bits in the learned schema provide us a meassure of the incompleteness of that model.

## 2.3 The sampling space

In this work the learning examples are noiseless but partial observations of the execution of a temporal plan starting from a given initial state. This means that if the value of a state variable is observed then, that is the actual value of that variable but that not the value of all the state variables can be observed at any time. For instance, just a subset of them is observable because there are associated sensors reporting their value.

PDDL2.2 is an extension of PDDL2.1 that includes the notion of *Timed Initial Literal* [12] ($\mathsf{til}(f, t)$), as a way of representing that a given Boolean variable $f \in F$ becomes true at a certain time $t > 0$, independently of the actions in the plan. Traditionally TILs are useful to model *exogenous happenings*; for instance, a time window when a warehouse is open in a logistics scenario, $\mathsf{til}(open, 8)$ and $\mathsf{til}(\neg open, 20)$. In this work we show that TILs are also suitable tool to model the observation of plan execution. The only difference with respect to the original semantic of TILs is that now, as happens with goals, observations represent conditions that must be *supported* by the execution of the plan at a particular time.

Figure 2 shows an example of the obsevation of a plan execution that is taken from the *driverlog* domain.

[1] Note that 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 unusual

**Figure 2.** Example of the obsevation of a plan execution.

## 3 Learning Action Models

This section formalizes the learning task we address in this paper and presents our CP formulation for addressing it with off-the-shelf CSP solvers.

### 3.1 One-shot learning of concurrent action models

We define the task of the *one-shot learning of temporal action models* as a tuple $\langle F, I, G, A?, O, C\rangle$, where:

- $\langle F, I, G, A?\rangle$ is a *temporal planning problem* where actions in $A?$ are partially specified (i.e., the exact *conditions/effects* and/or the *duration* of actions are unknown). In the worst case, we only know the vocabulary of the symbols that can appear in the *conditions/effects* of the actions. Available prior knowledge can be used to bound this vocabulary for certain action schemes.
- $O$ is the *observation* of a plan execution. Al least it contains a full observation of the initial state (time-stamped with $t = 0$) and a final state observation that equals the goals $G$ of the given *temporal planning problem* and it is time stamped with $t_{end}$, the maximum makes-span of a plan that solves that planning problem. Additionally, it can also contain time-stamped observations of the traversed intermediate states and about the time when actions started/ended their execution.
- $C$ is a set of *state-constraints* that reflects domain-specific expert knowledge. These constraints allow us to complete the input observation and/or to prune inconsistent action models. Figure 3 show an example of a set of state-constraints for the *driverlog* domain.

$$\forall x_1, y_1, y_2\ \neg at(x_1, y_1) \vee \neg at(x_1, y_2), \neq (y_1, y_2).$$
$$\forall x_1, y_1, y_2\ \neg in(x_1, y_1) \vee \neg in(x_1, y_2), \neq (y_1, y_2).$$
$$\forall x_1, y_1, y_2\ \neg driving(x_1, y_1) \vee \neg driving(x_1, y_2), \neq (y_1, y_2).$$
$$\forall x_1, y_1, y_2\ \neg driving(y_1, x_1) \vee \neg driving(y_2, x_1), \neq (y_1, y_2).$$
$$\forall x_1, y_1, y_2\ \neg at(x_1, y_1) \vee \neg driving(x_1, y_2).$$
$$\forall x_1, y_1, y_2\ \neg in(x_1, y_1) \vee \neg driving(x_1, y_2).$$
$$\forall x_1, y_1, y_2\ \neg at(x_1, y_1) \vee \neg in(x_1, y_2).$$
$$\forall x_1, y_1\ \neg empty(x_1) \vee \neg driving(y_1, x_1).$$
$$\forall x_1, y_1\ \neg empty(x_1) \vee \neg driving(y_1, x_1).$$
$$\forall x_1\ \neg link(x_1, x_1).$$
$$\forall x_1\ \neg path(x_1, x_1).$$

**Figure 3.** Examples of state-constraints for the *driverlog* domain.

A *solution* to the *one-shot learning of temporal action models* is a fully specified model of temporal actions $\mathcal{A}$ such that: (1), the conditions, effects and duration of the actions in $\mathcal{A}$ is completely specified, (there is no uncertainty about them). (2), the specified conditions, effects and duration of the actions in $\mathcal{A}$ are *consistent* with the given inputs $\langle F, I, G, A?, O, C\rangle$. In other words, such that we can build on top of the actions in $\mathcal{A}$ a valid plan whose execution starts in $I$, can produce the observations in $O$, and that reaches a final state that satisfies $G$.

### 3.2 Constraint satisfaction for learning action models

Given a *one-shot learning* task, defined as in subsection 3.1, our approach is to create a CSP whose solution induces an action model that solves the given *one-shot learning* task. Our CP formulation is solver-independent (this means that any off-the-shelf CSP solver that supports the expressiveness of our CP formulation can be used) and it is inspired by previous work on *temporal planning as CP* [19, 8].

#### 3.2.1 The CSP Variables

For each action $a$ in $A?$, the CSP contains the seven kinds of variables specified in Table 1. For simplicity, we model time in $\mathbb{Z}^+$ and bound all maximum times to the makespan $t_{end}$ of the observed plan execution. If the observation of the plan makespan it is not available we consider a long enough domain for durations.

| **Variable** | **Domain** | **Description** |
|---|---|---|
| $\mathsf{start}(a)$ | *Known/derived value* | Start time for $a$. |
| $\mathsf{end}(a)$ | *Known/derived value* | End time for $a$. |
| $\mathsf{dur}(a)$ | $[1..max(a)]$ | Duration of $a$ where $max(a) = t_{end} - \mathsf{start}(a)$. |
| $\mathsf{req\_start}(p, a)$, | $[0..t_{end}]$ | Interval $[\mathsf{req\_start}(p, a)..\mathsf{req\_end}(p, a)]$ |
| $\mathsf{req\_end}(p, a)$ | $[0..t_{end}]$ | during which action $a$ requires $p$. |
| $\mathsf{time}(p, a)$ | $[0..t_{end}]$ | Time when the effect $p$ of $a$ happens. |
| $\mathsf{sup}(p, a)$ | The action space | There is a causal link $\langle b_i, p, a\rangle$ s.t. $b_i$ is an action in the action space. |

**Table 1.** The CSP variables, their domains and semantics.

The value of the CSP variables $\mathsf{start}(a)$, $\mathsf{dur}(a)$ and $\mathsf{end}(a)$ is either given by the observation $O$ or derived from the expression $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. The remaining CSP variables model, respectively, the interval when conditions must hold, the time when the effects happen and the causal links of a solution plan that induces an action model that solves the given *one-shot learning* task.

Besides the actions of the given planning problem, we create two additional *dummy* actions:

- $\mathsf{init}$, that represents the *initial state* ($\mathsf{start}(\mathsf{init}) = 0$ and $\mathsf{dur}(\mathsf{init}) = 0$). This dummy action has no conditions so it has no associated variables $\mathsf{sup}$, $\mathsf{req\_start}$ and $\mathsf{req\_end}$ and has as many $\mathsf{time}(p_i, \mathsf{init}) = 0$ associated variables as $p_i$ in $I$.
- $\mathsf{goal}$, that represents the *last state observation* ($\mathsf{start}(\mathsf{goal}) = t_{end}$ and $\mathsf{dur}(\mathsf{goal}) = 0$). This dummy action has no effects so it has no $\mathsf{time}(p, a)$ variables and has as many associated variable $\mathsf{sup}(p_i, \mathsf{goal})$ and $\mathsf{req\_start}(p_i, \mathsf{goal}) = \mathsf{req\_end}(p_i, \mathsf{goal}) = t_{end}$ as different $p_i$ are in $G$.

Furthermore this formulation models TILs like any other regular actions. A $\mathsf{til}(f, t)$ can be seen as an additional *dummy* action ($\mathsf{start}(\mathsf{til}(f, t)) = t$ and $\mathsf{dur}(\mathsf{til}(f, t)) = 0$) with no conditions and the single effect $f$ that happens at time $t$ ($\mathsf{time}(f, \mathsf{til}(f, t)) = t$). The modeling of TILs is then analogous to init, as they both represent information that is given at a particular time, but externally to the execution of the plan. The formulation can model also time-stamped state observations (as TILS that must be *supported* by the execution of the solution plan). In other words as regular actions but with a fixed starting time.

### 3.2.2 *The CSP Constraints*

Table 2 shows the constraints defined among the CSP variables of Table 1. The first three constraints are explicit enough. The fourth constraint models *causal links* $\langle b, p, a\rangle$ (i.e., the time when $b$ supports $p$ must be before $a$ requires $p$). Note that in a causal link $\langle b, p, a\rangle$, $\mathsf{time}(p, b) < \mathsf{req\_start}(p, a)$ and not $\leq$ because, like in PDDL [6], our temporal planning model assumes an $\epsilon > 0$ ($\epsilon = 1$ since we are modeling time in $\mathbb{Z}^+$) as a small tolerance between the time when a given effect $p$ is supported and when it is required.

| ID | Constraint | Description |
|---|---|---|
| 1 | $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$ | End time of $a$. |
| 2 | $\mathsf{end}(a) \leq \mathsf{start}(\mathsf{goal})$ | Always goal is the last action of the plan. |
| 3 | $\mathsf{req\_start}(p, a) \leq \mathsf{req\_end}(p, a)$ | $[\mathsf{req\_start}(p, a)..\mathsf{req\_end}(p, a)]$ is a valid interval. |
| 4 | if $\mathsf{sup}(p, a) = b$ then<br>$\mathsf{time}(p, b) < \mathsf{req\_start}(p, a)$ | Modeling causal links $\langle b, p, a\rangle$. |
| 5 | $\forall c \neq a$ that deletes $p$ at time $t$:<br>if $\mathsf{sup}(p, a) = b$ then<br>$t < \mathsf{time}(p, b)$ OR $t > \mathsf{req\_end}(p, a)$ | Solving threat of $c$ to causal link $\langle b, p, a\rangle$ by promotion or demotion. |
| 6 | if $a$ requires and deletes $p$:<br>$\mathsf{time}(not - p, a) \geq \mathsf{req\_end}(p, a)$ | When $a$ requires and deletes $p$, the effect cannot happen before the condition. |
| 7 | $\forall a_i, a_j \mid a_i$ supports $p$ and<br>$a_j$ deletes $p$:<br>$\mathsf{time}(p, a_i) \neq \mathsf{time}(not - p, a_j)$ | Solving effect interference ($p$ and $not - p$) they cannot happen at the same time. |

**Table 2.** The CSP constraints and their semantics.

The fifth constraint avoids *threats* via *promotion* or *demotion* [11]. The sixth constraint models the fact that when the same action requires and deletes $p$ then the effect cannot happen before the condition. Note the $\geq$ inequality here; if one condition and one effect of $a$ happen at the same time, the underlying semantics in planning considers the condition is checked instantly before the effect [6]. The seventh constraint deals with the fact that two (possibly equal) actions have contradictory effects. These constraints apply to any type of action, including the additional dummy actions for representing $\mathsf{init}$, $\mathsf{goal}$, the TILs and the time-stamped state observations that are as given input of the *one-shot learning* task.

In addition to the constraints of Table 2 we can add input *state-constraint* (if available), like the ones pictured in Figure 3, to our CSP formulation to prune inconsistent action models.

### 3.2.3 *The CSP heuristics*

Our CSP formulation is solver-independent, which means we do not use heuristics that require changes in the implementation of the CSP engine. Although this reduces the solver performance, we are interested in using it as a blackbox that can be easily changed with no modification in our formulation. However, the experimentation showed us that the following *value selection* heuristics are effective to solve the defined CSPs:

1. $\mathsf{dur}(a)$, *lower values first*, thus preferring shortest solutions that make the learned model consistent.
2. $\mathsf{req\_start}(p, a)$ and $\mathsf{req\_end}(p, a)$. For $\mathsf{req\_start}$, lower values first, whereas for $\mathsf{req\_end}$, upper values first. This gives priority to $\mathsf{cond}_o(a)$, keeping conditions active as long as possible.
3. $\mathsf{time}(p, a)$. Lower values first, for negative effects, while upper values first, for positive effects. This gives priority to $\mathsf{eff}_s(a)$ delete effects and $\mathsf{eff}_e(a)$ positive effects.
4. $\mathsf{sup}(p, a)$, lower values first to prefer supporters that start earlier in the plan.

## 3.3 Specific constraints for the PDDL2.1 model

The defined CSP formulation deals with a *temporal planning* model that is more expressive than the defined by the PDDL2.1 language. For instance, it allows conditions and effects to happen at any time, even outside the execution of the action. 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 $\mathsf{req\_end}(p, a) = \mathsf{start}(a); \mathsf{req\_end}(p, a) = \mathsf{req\_start}(p, a) + 5$ fits this CSP formulation. Likewise the CSP formulation supports also the modeling of effects $p$ that happen in the middle of an action, e.g., $\mathsf{time}(p, a) = \mathsf{start}(a) + (\mathsf{dur}(a)/2)$.

We show here that, by adding extra constraints to our CSP formulation, we can make it PDDL2.1-compliant. Table 3 summarizes these extra constraints. To limit conditions to only be *at start*, *over all* or *at end* of an action execution we add constraints 1 and 2. Likewise, we make effects to exclusively happen either*at start* or *at end* of action executions with constraint 3. 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*: $\sum_{i=1}^{n=|\mathsf{eff}(a)|} \mathsf{time}(p_i, a) > n \cdot \mathsf{start}(a)$, which guarantees $\mathsf{eff}_e(a)$ is not empty.

| ID | Constraint | Description |
|---|---|---|
| 1 | $\mathsf{req\_start}(p, a) = \mathsf{start}(a)$ OR $\mathsf{req\_start}(p, a) = \mathsf{end}(a)$ | Conditions at start. |
| 2 | $\mathsf{req\_end}(p, a) = \mathsf{start}(a)$ OR $\mathsf{req\_end}(p, a) = \mathsf{end}(a)$ | Conditions at end. |
| 3 | $\mathsf{time}(p, a) = \mathsf{start}(a)$ OR $\mathsf{time}(p, a) = \mathsf{end}(a)$ | Effects at start or at end. |
| 4 | $\sum_{i=1}^{n=\lvert\mathsf{eff}(a)\rvert} \mathsf{time}(p_i, a) > n \cdot \mathsf{start}(a)$ | At least one effect. |
| 5 | $\forall a_i, a_j$ instances of the same operator:<br>$\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$ | Duration of the schema instantiations. |
| 7 | $\forall p_i : (\forall a_j : \mathsf{req\_start}(p_i, a_j) = \mathsf{start}(a_j))$ OR<br>$(\forall a_j : \mathsf{req\_start}(p_i, a_j) = \mathsf{end}(a_j))$ | Conditions of the schema instantiations. |
| 8 | $\forall p_i : (\forall a_j : \mathsf{req\_end}(p_i, a_j) = \mathsf{start}(a_j))$ OR<br>$(\forall a_j : \mathsf{req\_end}(p_i, a_j) = \mathsf{end}(a_j))$ | |
| 9 | $\forall p_i : (\forall a_j : \mathsf{time}(p_i, a_j) = \mathsf{start}(a_j))$ OR<br>$(\forall a_j : \mathsf{time}(p_i, a_j) = \mathsf{end}(a_j))$ | Effects of the schema instantiations. |

**Table 3.** The CSP constraints for the PDDL2.1.

Note that if a condition is never deleted in a plan, it can be considered an *invariant* condition for such a plan that represents *static* knowledge; e.g. a link between two locations that makes driving possible, or modeling a petrol station that allows a refuel action in a given location, etc. The constraint to be added for *invariant conditions* $p \in \mathsf{cond}_o(a)$ is simply: $((\mathsf{req\_start}(p, a) = \mathsf{start}(a))$ AND $(\mathsf{req\_end}(p, a) = \mathsf{end}(a)))$, i.e. Surprisingly, invariant conditions are modeled differently depending on the human modeler. See, for instance, `(link ?from ?to)` of Fig. 1, 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 can be planned to delete 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 temporal planning tasks depends on the human's *common sense*. On the contrary, our formulation checks the invariant conditions and deals with them always in a consistent way.

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, constraints 6,7 and 8 guarantee a constant structure for all the instances of the same operator.

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. 1 will last 2 time units no matter its parameters. 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. We model both cases adding constraint 5.

# 4 A unified formulation for planning, validation and learning

This section shows that our formulation for the *one-shot learning of temporal action models* is connected to the tasks of plan *synthesis* and plan *validation* and that this connection applies not only to temporal planning but also to the classical planning model, the vanilla model of AI planning where actions are instantaneous [9].

The connection between the planning, validation and learning tasks lies on the fact that we can constrain the CSP variables that model the conditions and effects of actions to known values. This feature is usefull to leverage a priori knowledge of a given planning domain. For instance, because we have some knowledge about the possible durations of a given action or because we already know that a given action produces for sure certain effects. This approach allows to synthesize a plan with a given action model, in this case every variable representing the conditions, effects and duration of the actions are constrained to a single value. Likewise we can validate a plan contrained the value of the variales representing the time-stamps for actions starting time, as we will see in section 5.

What is more, we can either synthesize (or validate) a plan despite some of the variables that representing the conditions, effects or duration of an action do not have a fixed value (its value is a priori unknwon). When addressing learning, planning or validating tasks, our formulation is flexible to accept different levels of specificatoin of the input knowledge:

- Partial knowledge of the conditions/effects of actions.
- Partial knowledge of action durations (i.e. a set of possible durations).
- Partial knowledge of the plan to validadate or synthesize.

To illustrate this, 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 to the CSP 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. We can also represent known values for the durations by bounnding the value of $\mathsf{dur}(a)$ variables to a given value. We can also introduce a priori knowledge about plans by bounding the value of the $\mathsf{start}(a)$ variables.

Last but not least, our CSP formulation can be adapted straightforward to address learning, planning or validation tasks within the classical planning model. In this case actions cannot have conditions *overall* or *at end* as well as they cannot have *at start* effects. Therefore the variables representing this kind of information can be removed from the CSP model (or be set to `false`). Further the duration of any action is fixed to one unit [14].

# 5 Evaluation

The CP formulation has been implemented in Choco[2], an open-source Java library for constraint programming that provides an object-oriented API to state the constraints to be satisfied.

The empirical evaluation of a learning task can be addressed from two perspectives. From a pure syntactic perspective, learning can be considered as an automated design task to create a new model that is similar to a reference (or *ground truth*) model. Consequently, the success of learning is an accuracy measure of how similar these two models are, which usually counts the number of differences (in terms of incorrect durations or distribution of conditions/effects). Unfortunately, there is not a unique reference model when learning temporal models at real-world problems. Also, a pure syntax-based measure usually returns misleading and pessimistic results, as it may count as incorrect a different duration or a change in the distribution of conditions/effects that really represent equivalent reformulations of the reference model. For instance, given the example of Fig. 1, the condition learned `(over all (link ?from ?to))` would be counted as a difference in action `drive-truck`, as it is `at start` in the reference model; but it is, semantically speaking, even more correct. Analogously, some durations may differ from the reference model but they should not be counted as incorrect. As seen in section **??**, some learned durations cannot be granted, but the underlying model is still consistent. Therefore, performing a syntactic evaluation in learning is not always a good idea.

From a semantic perspective, learning can be considered as a classification task where we first learn a model from a training dataset, then tune the model on a validation test and, finally, asses the model on a test dataset. Our approach represents a one-shot learning task because we only use one plan sample to learn the model and no validation step is required. Therefore, the success of the learned model can be assessed by analyzing the success ratio of the learned model *vs.* all the unseen samples of a test dataset. In other words, we are interested in learning a model that fits as many samples of the test dataset as possible. This is the evaluation that we consider most valuable for learning, and define the success ratio as the percentage of samples of the test dataset that are consistent with the learned model. A higher ratio means that the learned model explains, or adequately fits, the observed constraints the test dataset imposes.

## 5.1 Learning from partially specified action models

We have run experiments on nine IPC planning domains. It is important to highlight that these domains are encoded in PDDL2.1, with the number of operators shown in Table 4, so we have included the constraints given in section 3.3. We first get the plans for these domains by using five planners (*LPG-Quality* [10], *LPG-Speed* [10], *TP* [14], *TFD* [5] and *TFLAP* [17]), where the planning time is limited to 100s. The actions and observations on each plan are automatically compiled into a CSP learning instance. Then, we run the one-shot learning task to get a temporal action model for each instance, where the learning time is limited to 100s on an Intel i5-6400 @ 2.70GHz with 8GB of RAM. In order to assess the quality of the learned model, we 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. For instance, the *zenotravel* domain contains 78 instances, which means learning 78 models. Each model is validated by using the 77 remaining models, thus producing 78×77=6006 validations

[2] `http://www.choco-solver.org`

per struct, dur and struct+dur each. The value for each cell is the average success ratio. 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 validations), 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 structure or duration that leads to inconsistency in a sample counts as a failure. As seen in Table 4, the results are specially good, taking into consideration that we use only one sample to learn the temporal action model. These 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.

| | **ops** | **ins** | **struct** | **dur** | **struct+dur** |
|---|---|---|---|---|---|
| *zenotravel* | 5 | 78 | 100% | 68.83% | 35.76% |
| *driverlog* | 6 | 73 | 97.60% | 44.86% | 21.04% |
| *depots* | 5 | 64 | 55.41% | 76.22% | 23.19% |
| *rovers* | 9 | 84 | 78.84% | 5.35% | 0.17% |
| *satellite* | 5 | 84 | 80.74% | 57.13% | 40.53% |
| *storage* | 5 | 69 | 58.08% | 70.10% | 38.36% |
| *floortile* | 7 | 17 | 100% | 80.88% | 48.90% |
| *parking* | 4 | 49 | 86.69% | 81.38% | 54.89% |
| *sokoban* | 3 | 51 | 100% | 87.25% | 79.96% |

**Table 4.** Number of operators to learn. Instances used for validation. Average success ratio of the one-shot learned model *vs.* the test dataset in different IPC planning domains.

We have observed that some planners return plans with unnecessary actions, which has a negative impact for learning precise durations. 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 parallel actions for taking pictures/samples and navigation of multiple rovers, learning the duration and the structure+duration is particularly complex in this domain.

## 5.2 Learning from scratch

## 6 Conclussions

We have presented a purely declarative CP formulation, which is independent of any CSP solver, to address the learning of temporal action models. Learning in planning is specially interesting to recognize past behavior in order to predict and anticipate actions to improve decisions. The main contribution is a simple formulation that is automatically derived from the actions and observations on each plan execution, without the necessity of specific hand-coded domain knowledge. It is also flexible to support a very expressive temporal planning model, though 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. This reduces both the size of the required datasets and the computation time. The one-shot learned models are very good 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 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. According to 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, and even impossible, when many actions are executed in parallel.

Finally, our CP formulation can be represented and solved by Satisfiability Modulo Theories, which is part of our current work. As future work, we want to extend our formulation to learn meta-models, as combinations of many learned models, and a more complete action model. In the latter, rather than using a partially specified set of actions, we want to find out the conditions/effects together with their distribution. The underlying idea of finding an action model consistent with all the constraints will remain the same, but the model will need to be extended with additional decision variables and constraints. This will probably lead to the analysis of new heuristics for resolution.

## REFERENCES

[1] Eyal Amir and Allen Chang, 'Learning partially observable deterministic action models', *Journal of Artificial Intelligence Research*, **33**, 349–402, (2008).

[2] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty, 'A review of learning planning action models', *The Knowledge Engineering Review*, **33**, (2018).

[3] S. N. Cresswell, T.L. McCluskey, and M.M West, 'Acquiring planning domain models using LOCM', *The Knowledge Engineering Review*, **28(2)**, 195–213, (2013).

[4] William Cushing, Subbarao Kambhampati, Daniel S Weld, et al., 'When is temporal planning really temporal?', in *Proceedings of the 20th international joint conference on Artifical intelligence*, pp. 1852–1859. Morgan Kaufmann Publishers Inc., (2007).

[5] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger, 'Using the context-enhanced additive heuristic for temporal and numeric planning', in *Nineteenth International Conference on Automated Planning and Scheduling*, (2009).

[6] Maria Fox and Derek Long, 'Pddl2.1: An extension to pddl for expressing temporal planning domains', *Journal of artificial intelligence research*, **20**, 61–124, (2003).

[7] Daniel Furelos Blanco, Antonio Bucchiarone, and Anders Jonsson, 'Carpool: Collective adaptation using concurrent planning', in *AAMAS 2018. 17th International Conference on Autonomous Agents and Multiagent Systems; 2018 Jul 10-15; Stockholm, Sweden.[Richland]: IFAAMAS; 2018.* International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), (2018).

[8] Antonio Garrido, Marlene Arangu, and Eva Onaindia, 'A constraint programming formulation for planning: from plan scheduling to plan generation', *Journal of Scheduling*, **12**(3), 227–256, (2009).

[9] Hector Geffner and Blai Bonet, 'A concise introduction to models and methods for automated planning', *Synthesis Lectures on Artificial Intelligence and Machine Learning*, **8**(1), 1–141, (2013).

[10] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina, 'Planning through stochastic local search and temporal action graphs in lpg', *Journal of Artificial Intelligence Research*, **20**, 239–290, (2003).

[11] Malik Ghallab, Dana Nau, and Paolo Traverso, *Automated Planning: theory and practice*, Elsevier, 2004.

[12] J. Hoffmann and S. Edelkamp, 'The deterministic part of IPC-4: an overview', *Journal of Artificial Intelligence Research*, **24**, 519–579, (2005).

[13] Richard Howey, Derek Long, and Maria Fox, 'VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL', in *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, pp. 294–301. IEEE, (2004).

[14] Sergio Jiménez, Anders Jonsson, and Héctor Palacios, 'Temporal planning with required concurrency using classical planning', in *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, (2015).

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

[16] Jirí Kucera and Roman Barták, 'LOUGA: learning planning operators using genetic algorithms', in *Pacific Rim Knowledge Acquisition Workshop, PKAW-18*, pp. 124–138, (2018).

[17] Eliseo Marzal, Laura Sebastia, and Eva Onaindia, 'Temporal landmark graphs for solving overconstrained planning problems', *Knowledge-Based Systems*, **106**, 14–25, (2016).

[18] Kira Mourão, Luke S. Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman, 'Learning STRIPS operators from noisy and incomplete observations', in *Conference on Uncertainty in Artificial Intelligence, UAI-12*, pp. 614–623, (2012).

[19] Vincent Vidal and Héctor Geffner, 'Branching and pruning: An optimal temporal pocl planner based on constraint programming', *Artificial Intelligence*, **170**(3), 298–335, (2006).

[20] Qiang Yang, Kangheng Wu, and Yunfei Jiang, 'Learning action models from plan examples using weighted MAX-SAT', *Artificial Intelligence*, **171**(2-3), 107–143, (2007).

[21] Hankz Hankui Zhuo and Subbarao Kambhampati, 'Action-model acquisition from noisy plan traces', in *International Joint Conference on Artificial Intelligence, IJCAI-13*, pp. 2444–2450, (2013).

[22] Hankz Hankui Zhuo and Subbarao Kambhampati, 'Model-lite planning: Case-based vs. model-based approaches', *Artificial Intelligence*, **246**, 1–21, (2017).

[23] Hankz Hankui Zhuo, Tuan Anh Nguyen, and Subbarao Kambhampati, 'Refining incomplete planning domain models through plan traces', in *International Joint Conference on Artificial Intelligence, IJCAI-13*, pp. 2451–2458, (2013).

[24] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li, 'Learning complex action models with quantifiers and logical implications', *Artificial Intelligence*, **174**(18), 1540–1569, (2010).