One-Shot Learning of Temporal Action Models with Constraint Programming

Antonio Garrido and Sergio Jiménez

Universitat Politècnica de València
Camino de Vera s/n. 46022 Valencia, Spain
{agarridot,serjice}@dsic.upv.es

Abstract. This work presents a *constraint programming* approach for the learning and validation of durative actions for temporal planning. This work analyses the extreme learning scenario where just a single partial observation of the execution of a temporal plan is available. Our approach assumes that the given input observation is partial but noiseless, meaning that if a value is observed, then the observation is correct. Further the given input observation may refer to the execution of concurrent actions which makes our approach suitable for learning in multi-agent environments.

Keywords: Learning action models · Temporal planning · Constraint programming.

1 Introduction

Automated planning is the model-based approach for the task of selecting the actions that achieve a given goal starting from a given initial state. *Classical planning* is considered the vanilla model for automated planning and it assumes: fully observable states under a deterministic world, instantaneous actions, and goal conditions that exclusively refer to the last state reached by a plan [6, 9].

Beyond classical planning, there is a bunch of planning models that relax the assumptions of classical planning to compute more detailed solutions than classical plans [9]. This work focuses on the *temporal planning* model that relaxes the assumption of *instantaneous actions* [4]. Actions in temporal planning are *durative*, which means that they have a duration and that the conditions/effects of actions must hold/happen at different times during the action execution. As a consequence, *durative actions* can be executed in parallel and overlap in several ways and *temporal plans* indicate the precise time-stamp when durative actions start and end [10, 2].

Despite the performance and potential of state-of-the-art planners, its applicability to real-world tasks is still somewhat limited because of the difficulty of specifying correct and complete planning models [13]. The more expressive the planning model is, the more evident becomes this knowledge acquisition bottleneck, which jeopardizes the usability of AI planning technology. This fact has led to a growing interest in the planning community for the learning of action

models [12, 1]. The objective of this learning task is to compute the actions' conditions and effects that are *consistent* with a set of observations (defined as some sequence of state changes, world transitions, expert demonstrations or plan traces/logs). Observation of past behavior provide indirect, but very valuable information to hypothesize the action models, thus helping future planning decisions and recommendations. Beyond plan synthesis the learning of action models from observations is also interesting for proactive assistants when recognizing activities of (human or software) agents to assist them in their daily activities and, as the last frontier, to 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 [14, 16–18]. Defining model learning as an optimization task over a set of observations does not guarantee correctness (the learned models may fail to explain all the observations). This paper analyzes the application of *Constraint Programming* (CP) for the *one-shot learning* of temporal action models, that is, the extreme case of learning durative action models from a single partial observation of the execution of a temporal plan. In this work we assume that the given input observation is partial but noiseless, meaning that if a value is observed, then the observation is correct.

Despite the learning of classical action models has previously been addressed by different approaches; to our knowledge none learns the temporal features. This involves: i) identifying how the action conditions and effects are temporally distributed within the action execution, and ii) estimate the action duration. Further, observations may refer to the execution of concurrent actions which makes our approach suitable for learning in multi-agent environments. Last but not least, our CP-based approach evidences that learning *durative* action models strongly resembles the task of synthesizing and validating temporal plans that satisfy a given set constraints imposed by the input observations.

As a motivating example, let us assume a logistics plan trace. Learning the temporal planning model from this single observation will allow us: i) to better understand the insights of the logistics scenario 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 action's 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 *classical* and *temporal* planning models that we follow in this work.

2.1 Classical Planning

Let F be a set of propositional variables (*fluents*). A *state* s is a full assignment of values to these variables so $|s| = |F|$ and the size of the state space is $2^{|F|}$.

A *classical planning problem* is 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 the states variables. We assume that the actions in A are grounded from given action schemas, as in the Planning Domain Definition Language (PDDL) [4, 8, 9]).

Each action $a \in A$ has a set of preconditions $\mathsf{pre}(a)$ and a set of effects $\mathsf{eff}(a)$; $\mathsf{pre}(a), \mathsf{eff}(a) \subseteq F$. 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. Fig. 1 shows an example of two action scheme for the actions of the *driverlog* domain taken from the International Planning Competition[1]. The schema `board-truck` is used to board 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.

```
(: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 representation of two action schemes from the *driverlog* domain.

In this work we define a *plan* π for a *classical planning problem* as an action sequence $\pi = (a_1, \ldots, a_n)$ that induces the state sequence $\langle s_1, s_2 \ldots s_n\rangle$, where each a_i is applicable in s_{i-1}, being $s_0 = I$, and generates state s_i. In every valid solution plan $G \subseteq s_n$, i.e. the goal condition is satisfied in the last state.

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 the classical planning model, but A represents now a set of *durative actions*.

There are several options that allow for a high expressiveness of durative actions. On the one hand, durative actions can have a fixed duration, a duration

[1] IPC, `http://www.icaps-conference.org/index.php/Main/Competitions`

that ranges within an interval or a distribution of durations. On the other hand, actions may have conditions/effects 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 [5].

A popular model for temporal planning is given by PDDL2.1 [4], a language that somewhat restricts temporal expressiveness, and that defines a durative action $a \in A$ with the following elments:

- $\mathsf{dur}(a)$, a positive value for the action duration.
- $\mathsf{cond}_s(a), \mathsf{cond}_o(a), \mathsf{cond}_e(a) \subseteq F$. Unlike the *pre*conditions of a classical action, now conditions must hold when a starts (*at start*), during the entire execution of a (*over all*) or when a finishes (*at end*), respectively. In the general 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 general case $\mathsf{eff}_s(a) \cup \mathsf{eff}_e(a) = \mathsf{eff}(a)$.

The semantics of a durative action $a \in A$ 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 schemes for durative actions that extend the classical schemes of Fig. 1 for temporal planning. Now `board-truck` has a fixed duration whereas in `drive-truck` the duration depends on the locations.

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 $s_{t_0} = I$ and $G \subseteq s_{t_n}$, being s_{t_n} 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.

3 One-Shot learning of temporal action models

We define the task of the one-shot learning of a temporal action model as a tuple $\langle F, I, G, A[\cdot], O \rangle$, where:

- $\langle F, I, G, A[\cdot] \rangle$ is a temporal planning problem in which actions are partially specified. This means that we do not know the exact structure of the actions in $A[\cdot]$ (in terms of the distribution of conditions/effects nor their duration)

[2] Note that in classical planning $pre(a) = \{p, \neg p\}$ is contradictory, but in temporal planning, $\mathsf{cond}_s(a) = \{p\}$ and $\mathsf{cond}_e(a) = \{\neg p\}$ is a possible, though unusual, situation

```
(: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. Schema for two durative actions from the *driverlog* domain.

but that we have some prior knowledge. In this work we assume that, for each action $a \in A[\cdot]$, we only know the sets $\mathsf{pre}(a) = (\mathsf{cond}_s(a) \cup \mathsf{cond}_o(a) \cup \mathsf{cond}_e(a))$ and $\mathsf{eff}(a) = (\mathsf{eff}_s(a) \cup \mathsf{eff}_e(a))$ (as this information can be extracted from the classical version of the planning problem).
- O is the single sequence of observations corresponding to a plan trace. This observation contains the time when each action in the plan started, i.e. all $\mathsf{start}(a)$ that have been observed (by a sensor or human observer).

A solution to this learning task is a fully specified model of temporal actions $\mathcal{A}$, where the duration and distribution of conditions/effects of all actions in $A[\cdot]$ is fully specified, as defined in section 2.2. Every action in $\mathcal{A}$ is consistent with the knowlege partially specified in $A[\cdot]$, can start as it is observed in O, and induces a valid temporal plan that when executed starting from I satisfies G.

Intuitively, $\mathcal{A}$ is a solution if it *explains* the input observation and its subjacent temporal model implies no contradictions in the states induced by their execution. This means that the partially specified model $A[\cdot]$ and the observations O impose constraints that the learned model $\mathcal{A}$ must satisfy. Consequently, formulating a CSP to address the learning task and finding a consistent solution that explains that formulation seems a suitable approach to learn temporal models.

3.1 Example

Given a partially specified model of actions $A[\cdot]$ and a single sequence of observations O, learning a temporal action model $\mathcal{A}$ may seem straightforward (as it *just* implies to distribute the conditions+effects in time and estimate durations). However, this is untrue.

	$\mathsf{pre}(a)$	$\mathsf{eff}(a)$	$\mathsf{start}(a)$
a1	$\mathsf{pre}(a1) = \{\}$	$\mathsf{eff}(a1) = \{p, r\}$	2
a2	$\mathsf{pre}(a2) = \{\}$	$\mathsf{eff}(a2) = \{p, q\}$	4
a3	$\mathsf{pre}(a3) = \{r\}$	$\mathsf{eff}(a3) = \{\neg p\}$	7
a4	$\mathsf{pre}(a4) = \{p, q, r\}$	$\mathsf{eff}(a4) = \{\neg r\}$	10

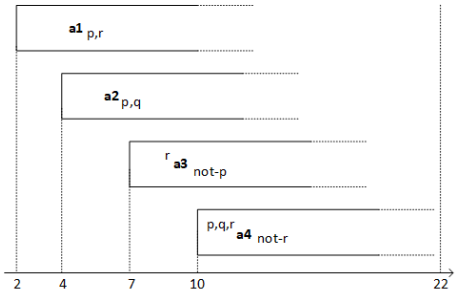

Fig. 3. An example of learning a temporal action model from $A[\cdot]$ and O. We know that the plan makespan is 22, but depending on the distribution of conditions, effects and durations, many configurations for actions a1,a2,a3 and a4 are possible.

Let us suppose the example of Fig. 3 where all the conditions, effects (but not their distributions) and start times of actions (but not their durations) are known, plus the plan makespan that is 22 in this case. 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. This shows that the distribution of the conditions and effects is relevant to the durations, and vice versa.

Action $a4$ needs p, which means two possible causal links ($\langle a1, p, a4\rangle$ or $\langle a2, p, a4\rangle$). The real causal link will be the last to happen, and 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 other supports happen; those supporting times and respective durations can never be assured. Therefore, in some situations the precise duration cannot be found and we can only provide values that make the model consistent.

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 $\neg p$ 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 $\mathsf{eff}_s(a2)$, this plan trace is inconsistent as $a3$ deletes p and no other action in the plan supports p for $a4$. However, if $\neg p$ 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; so is this a wrong decision of the human modeler? 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? 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. In Fig. 3, if the effects of $a1$ and $a2$ are *at start*, is it sensible to allow their durations to pass the limit of 22? In other words, once all plan goals are achieved, can the actions be executed beyond the plan makespan or they need to be cut off to such a value? This could potentially lead to an infinite number of models and overlapping situations, so it is not commonly accepted.

As can be noticed from above, learning a temporal action model is not straightforward, and many possible combinations are feasible provided they fit the constraints the model imposes.

4 A CP formulation for learning durative actions

Our approach is to create a single CSP that contains all the constraints imposed by the learning task. This includes: i) the observations on the start times; ii) the actions' conditions, effects and durations; and iii) the causal structure of the plan with the supports, to avoid threats and possible contradictory effects.

This formulation is inspired on previous work for the synthesis of temporal plans with CP[5]. The formulation is solver-independent which 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 The variables

For each action a in $A[\cdot]$, we create the seven kinds of variables specified in Table 1. These variables indicate the time-stamps 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[3].

This formulation is more expressive than PDDL2.1 (see more details in section 4.3 below) since it allows conditions and effects to be at any time, even

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

outside the execution of the action. For instance, 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.

Variable	Domain	Description
$\mathsf{start}(a)$	*known value*	start time of a observed in O
$\mathsf{dur}(a)$	$[1..makespan]$	duration of a. Optionally, it can be bounded by $makespan - \mathsf{start}(a)$
$\mathsf{end}(a)$	*derived value*	end time of a: $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$
$\mathsf{sup}(p,a)$	$\{b_i\}$ that supports p	Boolean 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

Table 1. Formulation of the variables and their domains for an action $a \in A[\cdot]$.

We create two dummy actions init and goal for each planning problem $\langle F, I, G, A[\cdot]\rangle$. First, init represents the initial state I ($\mathsf{start}(\mathsf{init}) = 0$ and $\mathsf{dur}(\mathsf{init}) = 0$). Action init has no variables sup, req_start and 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 $\mathsf{dur}(\mathsf{goal}) = 0$). Action 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.

4.2 The constraints

Table 2 shows the constraints that we define among the variables of Table 1. The three first constraints are trivial. 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 between the time when an effect p is supported and when it is required [4]. Since we model time in $\mathbb{Z}^+$, $\epsilon = 1$ and $\leq$ becomes $<$. The fifth constraint avoids any threat via promotion or demotion [9]. 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 a happen at the same time, the underlying semantics of planning considers the condition is checked instantly before the effect. The seventh constraint solves the fact that two (possible equal) actions have contradictory effects. It is important to note that constraints involve any type of action, including init and goal.

Constraint	Description
$\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$	end time of a
$\mathsf{end}(a) \leq \mathsf{start}(\mathsf{goal})$	goal is always the last action of the plan
$\mathsf{req\_start}(p,a) \leq \mathsf{req\_end}(p,a)$	$[\mathsf{req\_start}(p,a)..\mathsf{req\_end}(p,a)]$ 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 $b_j \neq a$
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 happens after 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

Table 2. Formulation of constraints.

4.3 Specific constraints for PDDL2.1 durative actions

As Section 2.2 explains, PDDL2.1 restricts the expressiveness of temporal planning in terms of conditions, effects, durations and structure of the actions. Our formulation is more expressive than PDDL2.1, but adding constraints to make it fully PDDL2.1-compliant is straightforward.

First, $((\mathsf{req\_start}(p,a) = \mathsf{start}(a))$ 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 is never deleted in a plan, it can be considered as an invariant condition for such a plan. In other words, it represents static information. This type of condition is commonly used in planning to ease the grounding process from the operators; e.g. to model that there is a link between two locations and, consequently, a driving is possible, or modeling a petrol station that allows a refuel action in a given location, etc. Therefore, the constraint to be added for an invariant condition p 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) no action can be planned to delete that link, and ii) the link should be necessary all over the driving. 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 a planning domain is not easy and it highly depends on 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*: $\sum_{i=1}^{n=|\mathsf{eff}(a)|} \mathsf{time}(p_i,a) > n \cdot \mathsf{start}(a)$, 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 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.

Four, 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 equal 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 $A[\cdot]$ 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.

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 $\mathsf{time}(\neg p, a3) \in [0..22]$. 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 $((\mathsf{time}(\neg r, a4) < \mathsf{time}(r, a1))$ OR $(\mathsf{time}(\neg r, a4) > \mathsf{req\_end}(r, a3)))$; $\mathsf{time}(\neg p, a3) \neq \mathsf{time}(p, a1)$ and $\mathsf{time}(\neg p, a3) \neq \mathsf{time}(p, a2)$.

There are many consistent solutions for this simple example, mainly because there is a huge range of possible durations that make the learned model consistent with the partially specified model $A[\cdot]$. Fig. 4 shows six arbitrary solutions. What is important to note is that the structure, i.e. distribution of conditions/effects, is similar in all the solutions. Actually, the distribution of the effects is identical except in solution 2 for q, 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_e). This shows that the one-shot learning, using only one sample, returns not only consistent models but also similar, which is very positive. The durations are, however, more different: $\mathsf{dur}(a1)$ ranges in the shown solutions from 7 to 19, whereas $\mathsf{dur}(a2)$ ranges from 5 to 18. As explained in section 3.1, learning the precise duration from just one sample may not be always possible, which is the main limitation of the one-shot learning task. Clearly, the duration learned for only one sample of `drive-truck(truck1,loc1,loc2,driver1)` cannot be always generalized to any other driving between these two locations. 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 as the learned duration must be consistent with all those occurrences.

Action	dur	cond_s	cond_o	cond_e	eff_s	eff_e
Solution 1						
$a1$	8				r	p
$a2$	18				q	p
$a3$	1		r			$\neg p$
$a4$	1		q, r	p		$\neg r$
Solution 2						
$a1$	19				r	p
$a2$	5					p, q
$a3$	1			r		$\neg p$
$a4$	1	r	p, q			$\neg r$
Solution 3						
$a1$	7				r	p
$a2$	18				q	p
$a3$	1			r		$\neg p$
$a4$	1		p, q, r			$\neg r$

Action	dur	cond_s	cond_o	cond_e	eff_s	eff_e
Solution 4						
$a1$	7				r	p
$a2$	9				q	p
$a3$	1			r		$\neg p$
$a4$	1		p, q, r			$\neg r$
Solution 5						
$a1$	9				r	p
$a2$	6				q	p
$a3$	1		r			$\neg p$
$a4$	1		q, r	p		$\neg r$
Solution 6						
$a1$	8				r	p
$a2$	16				q	p
$a3$	1	r				$\neg p$
$a4$	1		q, r	p		$\neg r$

Fig. 4. Six different solutions to the example of Fig. 3, but there are many more.

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[4], an open-source Java library for constraint programming 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 can reduce the CSP resolution performance, we are interested in using the solver 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 Validation

As seen above, adding constraints allows us to restrict the temporal expressiveness of the learned model. But 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, or we can figure out that an effect p always happens at end. Our CP formulation can include this by simply adding $\mathsf{dur}(a) = value$ and $\mathsf{time}(p, a) = \mathsf{end}(a)$, which is useful to enrich the partially specified actions in $A[\cdot]$ of the learning task.

In particular, the possibility of adding these constraints is very appealing when used for validating whether a partial action model is consistent, as we will see in section 5. Let us assume that the distribution of all (or just a few) conditions and/or effects is known and, consequently, represented in the learning task. If a learned model 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 of the durations. If a solution is found the durations are

[4] http://www.choco-solver.org

consistent, or inconsistent otherwise. Hence, we have (at least) 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. However, the third option checks whether a learned model can fit the given constraints, thus reproducing a strict validation task similar to [10].

5 Evaluation

This sections shows the empirical evaluation of our CP approach for the one-shot learning of temporal action models.

5.1 Domains

Our learning method is evaluated at eight different temporal planning domains, all taken from the IPC and encoded in PDDL2.1 (so we have included the constraints introduced in section 4.3). Table 3 reports the number of actions schema for each domain.

5.2 Evaluation metric

The evaluation of the learned action models can be addressed from a pure syntactic perspective, comparing the learned models to a *ground truth model*. The success of the learning is given by an accuracy metric that asseses how similar is a learned model wrt the ground truth (e.g. counting the number of differences in terms of incorrect conditions/effects and durations).

In real problems there is no *ground truth model*. Further, pure syntactic metrics are often too pesimist since they account as incorrect a different duration or distribution of conditions/effects that actually 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 in action `drive-truck` (as it is `at start` in the reference model), but it is, semantically speaking, 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.

Our approach for the evaluation is assesing the quality of the learned models with respect to its performance on a test dataset of unseen temporal planning problems. The success of learned models is assessed by analyzing the success ratio of a learned model in all temporal planning instances in the test dataset 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.

5.3 Setup

For each of the domains we first build a test data set of plans using these five planners (*LPG-Quality*, *LPG-Speed*[7], *TP* [11], *TFD* [3] and *TFLAP* [15]), where the planning time is limited to 100 seconds.

For each plan in the test data set we create the CP formulation and we run the one-shot learning task to get a temporal action model, where the learning time is limited to 100s on an Intel i5-6400 @ 2.70GHz with 8GB of RAM. To evaluate the quality of each learned model, we validate it *vs.* the other models *w.r.t.* the structure, the duration and the structure+duration. For instance, the *zenotravel* domain contains 78 instances, which means that we learn 78 models. Each model is validated by using the 77 remaining models, thus producing 78×77=6006 validations per struct, dur and struct+dur each. Table 3 reports the average success ratio.

5.4 Experimental results

In *zenotravel*, the struct value means the distribution of conditions/effects learned by using only one plan sample is consistent with all (100%) the samples used as dataset, which is the perfect value. The dur value means the durations learned explain 68.83% of the dataset. The struct+dur value means that the learned model explains entirely 35.76% of the samples. As seen in Table 3, the results are specially good, taking into consideration that we use only one sample to learn the temporal action model. 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 quality of the plans. 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 is particularly complex in this domain.

6 Conclusions through Related Work

[INCLUIR AQUI RELATED WORK]

We have presented a purely declarative CP formulation that is independent of any CSP solver to address the learning of temporal action models. 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. Unlike other approaches that need to learn from datasets with many samples, we perform one-shot learning. This reduces both the size of the required datasets and the computation time. The learned models are very good and explain a high

	No. operators	No. instances	struct	dur	struct+dur
Zenotravel	5	78	100%	68.83%	35.76%
Driverlog	6	73	97.6%	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%	70.1%	38.36%
Floortile	7				
Parking	4				
Sokoban	3				

Table 3. Success ratio of the one-shot learned model *vs.* the test dataset in different IPC planning domains.

number of samples in the datasets used for testing. Moreover, the same 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 planning 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, it is important to note that our formulation can be also solved by SATisfiability Modulo Theories, which is part of our current work. As future work, we want to extend our formulation to learn a more complete action model. Rather than using a partially specified set of actions in $A[\cdot]$, 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 constraints and boolean variables $\mathsf{is\_condition}(\mathsf{p},\mathsf{a})$, $\mathsf{is\_effect}(\mathsf{p},\mathsf{a})$ to decide whether p is a condition or effect of action a.

References

1. Arora, A., Fiorino, H., Pellier, D., Métivier, M., Pesty, S.: A review of learning planning action models. The Knowledge Engineering Review **33** (2018)
2. Cushing, W., Kambhampati, S., Weld, D.S., 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)
3. Eyerich, P., Mattmüller, R., Röger, G.: Using the context-enhanced additive heuristic for temporal and numeric planning. In: Nineteenth International Conference on Automated Planning and Scheduling (2009)
4. Fox, M., Long, D.: Pddl2. 1: An extension to pddl for expressing temporal planning domains. Journal of artificial intelligence research **20**, 61–124 (2003)

5. Garrido, A., Arangu, M., Onaindia, E.: A constraint programming formulation for planning: from plan scheduling to plan generation. Journal of Scheduling **12**(3), 227–256 (2009)
6. Geffner, H., Bonet, B.: A concise introduction to models and methods for automated planning. Synthesis Lectures on Artificial Intelligence and Machine Learning **8**(1), 1–141 (2013)
7. Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs in lpg. Journal of Artificial Intelligence Research **20**, 239–290 (2003)
8. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL - The Planning Domain Definition Language. AIPS-98 Planning Competition Committee (1998)
9. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: theory and practice. Elsevier (2004)
10. Howey, R., Long, D., Fox, M.: Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In: 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004). pp. 294–301 (2004)
11. Jiménez, S., Jonsson, A., Palacios, H.: Temporal planning with required concurrency using classical planning. In: Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS) (2015)
12. Jiménez, S., De la Rosa, T., Fernández, S., Fernández, F., Borrajo, D.: A review of machine learning for automated planning. The Knowledge Engineering Review **27**(4), 433–467 (2012)
13. Kambhampati, S.: 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). vol. 22(2), pp. 1601–1604 (2007)
14. Kucera, J., Barták, R.: LOUGA: learning planning operators using genetic algorithms. In: Pacific Rim Knowledge Acquisition Workshop, PKAW-18. pp. 124–138 (2018)
15. Marzal, E., Sebastia, L., Onaindia, E.: Temporal landmark graphs for solving over-constrained planning problems. Knowledge-Based Systems **106**, 14–25 (2016)
16. Mourão, K., Zettlemoyer, L.S., Petrick, R.P.A., Steedman, M.: Learning STRIPS operators from noisy and incomplete observations. In: Conference on Uncertainty in Artificial Intelligence, UAI-12. pp. 614–623 (2012)
17. Yang, Q., Wu, K., Jiang, Y.: Learning action models from plan examples using weighted MAX-SAT. Artificial Intelligence **171**(2-3), 107–143 (2007)
18. Zhuo, H.H., Kambhampati, S.: Action-model acquisition from noisy plan traces. In: International Joint Conference on Artificial Intelligence, IJCAI-13. pp. 2444–2450 (2013)