One-Shot Learning of Temporal Actions Models via Constraint Programming

Antonio Garrido and **Sergio Jiménez**

Abstract. We present a *Constraint Programming* (CP) formulation for learning temporal planning action models from the observation of a single plan execution (*one shot*). Inspired by the CSP approach to *temporal planning*, our CP formulation models *time-stamps* for states and actions, *causal-link* relationships, condition *threats* and effect *interferences*. This modeling evidences the connection between the tasks of *plan synthesis*, *plan validation* and *action model learning* in the temporal planning setting. Our CP formulation is solver-independent so off-the-shelf CSP solvers can be used for the resolution of any of these three tasks. The performance of our CP formulation is assessed when *learning* and *validating* action models at several temporal planning domains specified in the PDDL2.1 representation language.

1 INTRODUCTION

Temporal planning is an expressive planning model that relaxes the assumption of instantaneous actions of *classical planning* [10]. Actions in temporal planning are called *durative*, have an associated duration and, their conditions/effects may hold/happen at different times [7]. This means that *durative actions* can be executed in parallel and overlap in several different ways [4], and that valid solutions for temporal planning instances specify the precise time-stamp when each durative action starts and ends [16].

Despite the potential of state-of-the-art planners, their application to real world problems is still somewhat limited mainly because of the difficulty of specifying correct and complete planning models [19]. The more expressive the planning model, the more evident becomes this *knowledge acquisition bottleneck* that jeopardizes the usability of planning technology. There are however growing efforts in the planning community for the machine learning of action models from sequential plans: since pioneering learning systems like ARMS [25], we have seen systems able to learn action models with *quantifiers* [2, 31], from *noisy* actions or states [22, 27], from *null state information* [3], or from *incomplete* domain models [28, 30].

Most of the cited approaches for model learning are purely inductive and require large input datasets, e.g. hundreds of plan observations, to compute statistically significant models. With the aim of understanding better the connection between the learning of durative action models, *temporal planning* and the validation of temporal plans, this paper follows a radically different approach and studies the singular learning scenario where just the observation of a single plan execution (*one-shot*) is available. We leverage a solver-independent CP formulation that integrates the *learning* of temporal planning action models with the *synthesis* and the *validation* of temporal plans.

As far as we know this paper presents the first approach for learning action models for the *temporal planning* setting. 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 additionally: i) identifying how conditions and effects are temporally distributed within the actions, and ii) estimate the action duration. Further our approach allows the learning of action models from observations of plans with overlapping actions. This feature makes our approach appealing for learning action models from observations of multi-agent environments [8].

2 BACKGROUND

This section formalizes the *temporal planning* and the *constraint satisfaction* model that we follow in this work.

2.1 Temporal Planning

We assume that *states* are factored into a set F of Boolean variables. A state s is a time-stamped assignment of values to all the variables in F. A *temporal planning problem* is a tuple $P = \langle F, I, G, A \rangle$ where the *initial state* I is a fully observed state (i.e. a total assignment of the state variables $|I| = |F|$) 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*.

A *durative action* has an associated duration and may have conditions/effects on F at different times [9, 24]. To compactly represent temporal planning problems, we assume that the state variables in F are intantiations of a given set of predicates Ψ (like in the PDDL language [26]) and that durative actions in A are fully grounded from *action schemes* (also known as *operators*).

PDDL2.1 is the input representation language for the temporal track of the International Planning Competition (IPC) [7, 12]. 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. $\mathsf{cond}_s(a), \mathsf{cond}_o(a), \mathsf{cond}_e(a)$ representing the three types of 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.
3. $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$ represent the two types of action effects. In PDDL2.1, effects can happen *at start* or *at end* of action a respectively, and can be either positive or negative (i.e. asserting or retracting variables).

PDDL2.1 is a restricted temporal planning model that defines the semantics of a *durative action* a as two discrete events, $\mathsf{start}(a)$ and

$\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. This means that if 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 the closed 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). Figure 1 shows an example of two schemes for PDDL2.1 durative actions taken from the *driverlog* domain. The `board-truck` schema defines a fixed duration of two time units while the duration of `drive-truck` depends on the driving time associated to the two given locations.

```
(: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 of *durative actions* represented in PDDL2.1.

PDDL2.2 is an extension of the PDDL2.1 representation language that includes the notion of *Timed Initial Literal* [14], denoted as $\mathsf{til}(f,t)$, and representing that variable $f \in F$ becomes true at a certain time $t > 0$, independently of the actions in the plan [5]. TILs are useful to model *exogenous events*; for instance, in a logistics scenario, the 8h-20h time window when a warehouse is open can be modeled with these two timed initial literals: $\mathsf{til}(openWarehouse, 8)$ and $\mathsf{til}(\neg openWarehouse, 20)$.

A *temporal plan* is a set of pairs $\pi = \{(a_1, t_{a_1}), (a_2, t_{a_2}) \dots (a_n, t_{a_n})\}$. Each pair (a, t_a) contains a *durative action* a and the action *time-stamp* $t_a = \mathsf{start}(a)$. 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. A *solution* to a given temporal planning problem P is a *temporal plan* π such that its execution, starting from the corresponding initial state, eventually reaches a state that meets the goal conditions, $G \subseteq s_{end}$. A solution is *optimal* iff it minimizes the plan *makespan* (i.e., the maximum $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$ of any actions in the plan).

2.2 Constraint Satisfaction

A *Constraint Satisfaction Problem* (CSP) is a tuple $\langle X, D, C \rangle$, where X is a set of finite-domain *variables*, D represents the *domain* for each of these variables and C is a set of *constraints* among the variables in X that bound their possible values in D.

A *solution* to a CSP is an assignment of values to all the variables in X that is *consistent* with all the input constraints. Given a CSP there may be many different solutions to that problem, i.e., different variable assignments that are *consistent* with the input constraints.

A *cost-function* can be defined over the variables in X to specify user preferences about the space of possible solutions. Given a CSP and a cost-function, then an *optimal solution* is a full variable assignment that is consistent with the constraints of the CSP such that it also minimizes the value of the defined cost-function.

3 One-shot learning of temporal actions models

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

- $\langle F, I, G, A? \rangle$ is a *temporal planning problem* such that the actions in $A?$ are *incomplete*. By *incomplete* we mean that the exact conditions/effects, their temporal annotation, and the duration of actions are unknown while the actions *header* (i.e., the *name* and *parameters* of each action) is known. With this regard, we say that a fluent $f \in F$ is *candidate* to appear in the condition/effects of an action $a \in A?$ iff f appears in the set of FOL interpretations of the predicates that shape the fluents F over the action parameters $pars(a)$. We call this subset of candidate fluents the *alphabet* of an action a and is denoted as $\alpha(a)$. For instance, Figure 2 shows α(board-truck(driver1,truck1,loc1)), i.e., the *alphabet* of six *candidates* to appear in the conditions/effect of the ground action *board-truck(driver1,truck1,loc1)*.

```
;;;
;; Alphabet board-truck(driver1,truck1,loc1)

(at driver1 loc1) (at truck1 loc1)
(driving driver1 truck1)  (empty truck1)
(path loc1 loc1) (link loc1 loc1)
```

Figure 2. *Alphabet* for the ground action *board-truck(driver1,truck1,loc1)*.

- O is the set of *observations* over a single plan execution. Al least this set 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 temporal planning problem, time-stamped with t_{end} (the *makespan* of the observed plan). Additionally, it can contain time-stamped observations of traversed intermediate *partial states*[1] as well as the times when actions start and/or end their execution. Figure 3 shows an example of the observation of a plan execution taken from the *driverlog* domain.
- C is a set of *constraints* that captures domain-specific expert knowledge. In this work these constraints are of two kinds:
  - Constraints that specify that a given $f \in \alpha(a)$ is actually in the conditions/effects of action a. These constraints allow to represent partially specified action models [29]. For instance we may know in advance that the action `board-truck` requires the `driver` and the `truck` to be at the same location.

[1] In this work, not all variables can be observed at any time; that is, we deal with *partial observations* (e.g. just a subset of variables is observable by associated sensors). Observations are noiseless, which means that if a value is observed, that is the actual value of that variable.

```
(:objects driver1 driver2 - driver
          truck1 truck2 - truck
          package1 package2 - obj
          s0 s1 s2 p1-0 p1-2 - location)

(:init (at driver1 s2) (at driver2 s2) (at truck1 s0)
       (empty truck1) (at truck2 s0) (empty truck2)
       (at package1 s0) (at package2 s0)
       (path s1 p1-0) (path p1-0 s1) (path s0 p1-0)
       (path p1-0 s0) (path s1 p1-2) (path p1-2 s1)
       (path s2 p1-2) (path p1-2 s2)
       (link s0 s1) (link s1 s0) (link s0 s2) (link s2 s0)
       (link s2 s1) (link s1 s2))

(:observation :time-stamp 56
              (at driver1 s1) (at truck1 s1))

(:observation :time-stamp 78
              (at package1 s0) (at package2 s0))
```

Figure 3. Example of a set of three observations (containing the fully observed initial state and two time-stamped partial states) extracted from the execution of a plan from the *driverlog* domain.

– Mutually-exclusive (*mutex*) constraints that allow to (1), deduce new observations and (2), prune action models inconsistent with these constraints. Figure 4 shows an example of a set of five *mutex constraints* for the *driverlog* domain.

$\forall truck, driver:$ $\neg empty(truck) \vee \neg driving(driver, truck)$.
$\forall driver, loc_1, loc_2:$ $\neg at(driver, loc_1) \vee \neg at(driver, loc_2), \neq (loc_1, loc_2)$.
$\forall driver, truck1, truck2:$ $\neg driving(driver, truck_1) \vee \neg driving(driver, truck_2), \neq (truck_1, truck_2)$.
$\forall drvr1, drvr2, truck:$ $\neg driving(drvr_1, truck) \vee \neg driving(driver_2, truck), \neq (drvr_1, drvr_2)$.
$\forall drvr, location, truck:$ $\neg at(drvr, location) \vee \neg driving(drvr, truck)$.

Figure 4. Examples of five *mutex constraints* for the *driverlog* domain.

A *solution* to a learning task $\mathcal{L}$ is a fully specified model of durative actions $\mathcal{A}$ such that the *conditions*, *effects*, their temporal annotations and the *duration* of any action in $\mathcal{A}$ are: i) completely specified; and ii) *consistent* with $\mathcal{L} = \langle F, I, G, A?, O, C \rangle$. By *consistent* we mean that there exists a valid plan that exclusively contains actions in $\mathcal{A}$ and whose execution starting in I, produces all the observations in O at the associated time-stamps, while it satisfies all constraints in C, and reaches a final state that satisfies G.

4 Learning action models with CSPs

Given a *one-shot learning task* $\mathcal{L}$ as defined in Section 3, we automatically create a CSP, whose solution induces an action model that solves $\mathcal{L}$. This method is solver-independent and integrates previous work on *temporal planning* as satisfiability [24, 17, 9, 23].

4.1 The variables

For each action $a \in A?$ and candidate $f \in \alpha(a)$ to appear in the conditions/effects of a, we create the eight CSP variables of Table 1.

Variable X1 represents the time when an action *starts* (its time-stamp), X2 represents when the action *ends* and variable X3 represents the action duration. The value of X1,X2 and X3 can either be observed in O or derived from the expression $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. We model time in $\mathbb{Z}^+$ and bound all maximum times to the *makespan* of the observed plan (t_{end} if observed in O). If the observation of t_{end} is unavailable, we consider a large enough domain for time. Boolean variables X4/X5 model whether f is actually a condition/effect of action a. X6.1 and X6.2 define the closed interval throughout condition f must hold for the application of action a (provided $\mathsf{is\_cond}(f, a)$=*true*). X7 models a *causal link* representing that action b supports f that is required by a. If f is not a condition of a ($\mathsf{is\_cond}(f, a)$=*false*) then $\mathsf{sup}(f, a)=\emptyset$, representing an empty supporter. Last but not least, variable X8 models the time-stamp when effect f happens in a (provided $\mathsf{is\_eff}(f, a)$=*true*).

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

ID	Variable	Domain	Description
X1	$\mathsf{start}(a)$	$[0..t_{end}]$	*Start time* of action a
X2	$\mathsf{end}(a)$	$[0..t_{end}]$	*End time* of action a
X3	$\mathsf{dur}(a)$	$[0..t_{end}]$	*Duration* of action a
X4	$\mathsf{is\_cond}(f, a)$	$\{0, 1\}$	1 if f is a *condition* of a; 0 otherwise
X5	$\mathsf{is\_eff}(f, a)$	$\{0, 1\}$	1 if f is an *effect* of a; 0 otherwise
X6.1	$\mathsf{req\_start}(f, a)$,	$[0..t_{end}]$	Interval when action a requires f
X6.2	$\mathsf{req\_end}(f, a)$		
X7	$\mathsf{sup}(f, a)$	$\{b\}_{b \in A?} \cup \emptyset$	Supporters for causal link $\langle b, f, a \rangle$
X8	$\mathsf{time}(f, a)$	$[0..t_{end}]$	Time when the effect f of a happens

This simple formulation is powerful enough to model tils and *observations*. The intuition is that modeling a til is analogous to modeling the *initial state* of a planning task (both represent information that is given at a particular time but externally to the execution of the plan). Likewise modeling observations is analogous to modeling the goals of a planning task, as they both represent conditions that must be satisfied by the execution of the plan. On the one hand a $\mathsf{til}(f, t)$ is modeled as a *dummy* action that starts at time t and has instantaneous duration ($\mathsf{start}(\mathsf{til}(f, t)) = t$ and $\mathsf{dur}(\mathsf{til}(f, t)) = 0$) with no conditions and the single effect f that happens at time t ($\mathsf{is\_eff}(f, \mathsf{til}(f, t))$=*true* and $\mathsf{time}(f, \mathsf{til}(f, t)) = t$). On the other hand, an observation $\mathsf{obs}(f, t)$ is modeled as another *dummy* action that also starts at time t and has instantaneous duration ($\mathsf{start}(\mathsf{obs}(f, t)) = t$ and $\mathsf{dur}(\mathsf{obs}(f, t)) = 0$) but with only one condition f, which is the value observed for fact f ($\mathsf{is\_cond}(f, \mathsf{obs}(f, t))$=*true*, $\mathsf{sup}(f, \mathsf{obs}(f, t)) \neq \emptyset$ and $\mathsf{req\_start}(f, \mathsf{obs}(f, t)) = \mathsf{req\_end}(f, \mathsf{obs}(f, t)) = t$), and no effects at all. Observations can also refer to the *start* and *end* of an action, $\mathsf{obs}(is\_start(a), t)$ represents it was observed that action a starts at t while $\mathsf{obs}(is\_end(a), t)$ represents it was observed that action a ends at t.

4.2 The constraints

Table 2 shows the constraints defined among the CSP variables of Table 1. C1 models the duration of an action while C2 indicates that actions must end before t_{end}. C3 forces to have a well-defined $[\mathsf{req\_start}, \mathsf{req\_end}]$ interval, when the conditions of action a are required. C4 models that only action conditions require supporters and C5 models that the time when b supports f must be before a requires it because of the causal link $\langle b, f, a \rangle$[2]. Given a causal link $\langle b, f, a \rangle$, constraint C6 avoids *threats* of actions c deleting f (threats are solved via *promotion* or *demotion* [12]). C7 prevents action a being a supporter of f when $\mathsf{is\_eff}(f, a)$=*false*. Constraint C8 models the fact that when the same action requires and deletes f the effect cannot happen before the condition. Note the $\geq$ inequality here: if one condition and one effect of the same action happen at the same time, the underlying semantics in planning considers the condition

[2] $\mathsf{time}(f, b) < \mathsf{req\_start}(f, a)$ and not $\leq$ because our temporal planning model assumes $\epsilon > 0$ (ϵ denotes a small tolerance that implies no collision between the time when effect f is supported and when it is required, like in PDDL2.1 [7]). When time is modeled in $\mathbb{Z}^+$, $\epsilon = 1$ so $\leq$ becomes $<$.

Table 2. The CSP constraints and brief description.

ID	Constraint	Description
C1	$\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$	Relationship among start, end and duration of a
C2	$\mathsf{end}(a) \leq \mathsf{start}(\mathsf{goal})$	Always goal is the last action of the plan
C3	**if** ($\mathsf{is\_cond}(f, a)$=*true*) **then** $\mathsf{req\_start}(f, a) \leq \mathsf{req\_end}(f, a)$	$[\mathsf{req\_start}(f, a)..\mathsf{req\_end}(f, a)]$ is a valid interval
C4	**iff** ($\mathsf{is\_cond}(f, a)$=*false*) **then** $\mathsf{sup}(f, a) = \emptyset$	f is not a condition of $a \iff$ the supporter of f in a is $\emptyset$
C5	**if** ($\mathsf{is\_eff}(f, b)$=*true*) **AND** ($\mathsf{is\_cond}(f, a)$=*true*) **AND** ($\mathsf{sup}(f, a) = b$)) **then** $\mathsf{time}(f, b) < \mathsf{req\_start}(f, a)$	Modeling the causal link $\langle b, f, a\rangle$: supporting f before it is required (obviously $b \neq \emptyset$)
C6	**if** ($\mathsf{is\_eff}(f, b)$=*true*) **AND** ($\mathsf{is\_cond}(f, a)$=*true*) **AND** ($\mathsf{is\_eff}(\textit{not-}f, c)$=*true*) **AND** ($\mathsf{sup}(f, a) = b$) **AND** ($c \neq a$) **then** ($\mathsf{time}(\textit{not-}f, c) < \mathsf{time}(f, b)$) **OR** ($\mathsf{time}(\textit{not-}f, c) > \mathsf{req\_end}(f, a)$)	Solving threat of c to causal link $\langle b, f, a\rangle$ by promotion or demotion (obviously $b \neq \emptyset$)
C7	**if** ($\mathsf{is\_eff}(f, a)$=*false*) **then** $\forall b$ that requires f: $\mathsf{sup}(f, b) \neq a$	a cannot be a supporter of f for any other action b
C8	**if** ($\mathsf{is\_cond}(f, a)$=*true*) **AND** ($\mathsf{is\_eff}(\textit{not-}f, a)$=*true*) **then** $\mathsf{time}(\textit{not-}f, a) \geq \mathsf{req\_end}(f, a)$	a requires and deletes f: the condition holds before the effect
C9	**if** ($\mathsf{is\_eff}(f, b)$=*true*) **AND** ($\mathsf{is\_eff}(\textit{not-}f, c)$=*true*) **then** $\mathsf{time}(f, b) \neq \mathsf{time}(\textit{not-}f, c)$	Solving effect interference at the same time (f and *not-*f)
C10	$\sum \mathsf{is\_cond}(f_i, a) \geq 1$ **AND** $\sum \mathsf{is\_eff}(f_j, a) \geq 1$ **forall** condition f_i and effect f_j of a	Every non-dummy action has at least one condition/effect

is checked instantly before the effect [7]. C9 prevents two actions have contradictory effects and C10 forces actions to have at least one condition and one effect (C9 applies to any action, including dummy actions init, goal, til and obs, while C10 only applies to *non-dummy* actions).

4.2.1 Constraints for PDDL2.1

The presented CSP formulation accommodates a level of expressiveness beyond PDDL2.1 because it allows conditions/effects to be at any time, even outside the execution of the action. For example, it allows a condition $f \in \alpha(a)$ to hold in $\mathsf{start}(a)\pm 2$: $\mathsf{req\_start}(f, a) = \mathsf{start}(a) - 2$ and $\mathsf{req\_end}(f, a) = \mathsf{start}(a) + 2$. Likewise an effect $f \in \alpha(a)$ might also happen after the action ends e.g., $\mathsf{time}(f, a) = \mathsf{end}(a) + 2$.

Making our formulation *PDDL2.1-compliant* is straightforward, by adding the constraints of Table 3 for all *non-dummy* actions: C11 limits the *conditions* of an action to be only at *at start*, *over all* or *at end*. C12 limits the *effects* of an action to only happen *at start* or *at end*. In PDDL2.1 the structure of conditions/effects of all actions $\{a_j\}$ grounded from a particular operator are fixed. With this regard, C13 makes the conditions of all $\{a_j\}$ the same while C14 makes the effects of all $\{a_j\}$ the same. C15 makes the duration of all occurrences of the same action equal (if desired, this is not mandatory in PDDL2.1). Last but not least, C16 forces all actions to have at least one of its *n*-effects *at end*. Actions with only *at start* effects turn the value of the duration irrelevant besides they could exceed the plan makespan (this last constraint is not specific of PDDL2.1 but produces more rationale models for *durative actions*).

Table 3. Constraints to learn PDDL2.1-compliant action models.

ID	Constraint
C11.1	($\mathsf{req\_start}(f, a) = \mathsf{start}(a)$) **OR** ($\mathsf{req\_start}(f, a) = \mathsf{end}(a)$)
C11.2	($\mathsf{req\_end}(f, a) = \mathsf{start}(a)$) **OR** ($\mathsf{req\_end}(f, a) = \mathsf{end}(a)$)
C12	($\mathsf{time}(f, a) = \mathsf{start}(a)$) **OR** ($\mathsf{time}(f, a) = \mathsf{end}(a)$)
C13.1	$\forall f_i : (\forall a_j : \mathsf{req\_start}(f_i, a_j) = \mathsf{start}(a_j))$ **OR** $(\forall a_j : \mathsf{req\_start}(f_i, a_j) = \mathsf{end}(a_j))$
C13.2	$\forall f_i : (\forall a_j : \mathsf{req\_end}(f_i, a_j) = \mathsf{start}(a_j))$ **OR** $(\forall a_j : \mathsf{req\_end}(f_i, a_j) = \mathsf{end}(a_j))$
C14	$\forall f_i : (\forall a_j : \mathsf{time}(f_i, a_j) = \mathsf{start}(a_j))$ **OR** $(\forall a_j : \mathsf{time}(f_i, a_j) = \mathsf{end}(a_j))$
C15	$\forall a_i, a_j$ occurrences of the same action: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$
C16	$\sum_{i=1}^{n} \mathsf{time}(f_i, a) > n \times \mathsf{start}(a)$

4.2.2 Mutex constraints

Mutex-constraints can be exploited in a pre-proces step for completing the input observations of a *one-shot learning task* $\mathcal{L}$. The set of mutexes that is given as input to a learning task $\mathcal{L}$ allows to infer new information: if two Boolean variables $\langle f_i, f_j\rangle$ are mutex they cannot hold simultaneously. This means that if we observe f_i, then we can infer $\neg f_j$ (despite $\neg f_j$ was not actually observed). Likewise if we observe f_j, we can infer $\neg f_i$. This source of additional knowledge is specially relevant for correctly learning *negative effects* when there is a lack of input observations. Mutexes helps to fill this void inferring the observation of negated variables, which forces later to satisfy the *causal links* of negative variables.

Note that, in *temporal planning* models with *durative actions*, given a $\langle f_i, f_j\rangle$ mutex $\neg f_i$ does not necessarily implies f_j. See the effects `(not (at ?t ?l1))` and `(at ?t ?l2)` of action `drive-truck` of Figure 1 that respectively happen *at start* and *at end*. If `(at ?t ?l1)` and `(at ?t ?l2)` are mutex (as defined in Figure 4), the same truck cannot be in two locations simultaneously. It is valid however for a truck to be, for some time, at no location. This situation does not happen however in classical planning models where actions have instantaneous effects, so if $\langle f_i, f_j\rangle$ are mutex then f_i implies *not-*f_j and vice versa.

Dynamic observations can be created to exploit mutex constraints at any generated intermediate state (even if the intermediate state was not observed at all but was inferred by the CSP solver). A mutex $\langle f_i, f_j\rangle$ means that, immediately after a asserts f_i, we need to ensure the observation *not-*f_j. Furher, if $\mathsf{is\_eff}(f_i, a)$ takes the value *true*, then the next observation is added: $\mathsf{obs}(\textit{not-}f_j, \mathsf{time}(f_i, a) + \epsilon)$. The time of the observation cannot be just $\mathsf{time}(f_i, a)$, as we first need to assert f_i and one ϵ later observe *not-*f_j. Adding the variables and constraints for this new observation during the CSP search is trivial for *Dynamic CSPs* (DCSPs) [21]. Otherwise, we need to statically define a new type of observation $\mathsf{obs}(f_i, a, \textit{not-}f_j)$, where a supports f_i which is mutex with f_j and, consequently, we will need to observe *not-*f_j. The difference *w.r.t.* an original obs is two-fold: i) the observation time is now initially unknown, and ii) the observation will be activated or not according to the following constraints:

if ($\mathsf{is\_eff}(f_i, a)$=*true*) **then** ($\mathsf{start}(\mathsf{obs}(f_i, a, \textit{not-}f_j)) = \mathsf{time}(f_i, a) + \epsilon$) **AND** ($\mathsf{is\_cond}(\textit{not-}f_j, \mathsf{obs}(f_i, a, \textit{not-}f_j))$=*true*)
else $\mathsf{is\_cond}(\textit{not-}f_j, \mathsf{obs}(f_i, a, \textit{not-}f_j))$=*false*

4.3 The cost-function

There is often a set of possible action models that are *consistent* with the inputs of a *one-shot learning of temporal action models* task. A *cost-function* allows us to define user preferences among these action models to guide the CSP process towards the most desired solutions.

For instance the *conditions* of actions that are never deleted by any action, are hard to be learned with a pure satisfiabiliy aproach (e.g., the `(link ?l1 ?l2)` condition in the `drive-truck` action showed in the Figure 1). In general, this is an issue when learning action models in which *static predicates* appear in the actions *condi-*

tions [13]. We address this issue extending the CP formulation to not only deal with the satisfaction of constraints but also to prefer action models that support the input observations as *compactly* as possible. To prefer this kind of *compact* support of the input observations we define the following two positive functions:

ϕ_1 *No-dummy causal links*. This function counts the number of causal links created to support the provided observations with no-dummy actions. That is causal links $\langle b, f, a\rangle$ such that: $\mathsf{obs}(f,t)$ is an input observation and action $b \neq start$, i.e., the supporter is not the *start* dummy action.

ϕ_2 *Side-effects*. This function counts the number of effects that are added by the actions but that are not building any causal link. Formally, the number of $\mathsf{time}(f,b)$ such that there is no $b = \mathsf{sup}(f,a)$ for any action a.

Our aim is to compute solutions to the CSP that minimizes functions ϕ_1 (to prefer the creation of causal links with the given input knowledge) and ϕ_2 (to prefer action models that tie up loos ends). To achieve this we ask the CSP solve to *pareto minimize* functions ϕ_1 and ϕ_2.

5 Flexibility of the CSP approach

This section displays the flexibility of our formulation for different tasks in the *temporal* planning setting (and even in the *classical* planning setting) and for different levels of input knowledge that express different levels of prior knowledge.

5.1 Incomplete and complete action models

In between having in advance a consistent value for all of the CSPs variables and not having any, there is a *grayscale* of different levels of prior knowledge. Focusing this discussion on our CP formulation, variables X3, X4, X5, X6 and X8 from Table 1 represent the duration, the conditions and effects of a given action a and their distribution over time. If this information is *apriori* known for a given action a (e.g. because we are not learning from scratch but filling the gaps of an *incomplete* action model, as introduced in Section 3) then these variables are initially set to the given values so they can be propagated by the CSP reducing the CSP branching factor.

Encoding *durative action* models with *finite-domain* variables allows also to compute similarity metrics between action models (e.g., to compare a learned action model with respect to a reference model). Given $\alpha(a)$, the *alphabet* of *candidates* to appear in the conditions/effect of an action a, the size of its space of possible action models is $\mathcal{D} \times 2^{5|\alpha(a)|}$ where $\mathcal{D}$ is the number of different possible durations for a (i.e., the domain of X3). This means that the *conditions* and *effects* of an action a can be compactly coded by 5 bit-vectors, each of length $|\alpha(a)|$ (three vectors representing the conditions *at start*, *overall*, *at end* and two vectors representing the effects *at start* and *at end*). A 0-bit in the vector represents that the corresponding *condition/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 syntactic similarity metric between durative actions. In this case, the number of wrong 1-bits in the learned schema provide us a measure of the *incorrectness* of the learned model (number of *conditions* and *effects* that should not be in the learned model) and the number of wrong 0-bits in the learned schema provide us a measure of the *incompleteness* of that model (number of *conditions* and *effects* that are missing in the learned model). A similar method was already defined for *strips* action models [1].

5.2 Integrating planning, validation and learning

Our CSP formulation provides an integrative view for the tasks of plan *synthesis*, plan *validation* and the learning of action models in the *temporal planning* setting. This integration lies on the fact that, within the same CSP formluation, we can initially set some CSP variables to known values, when they are available.

In more detail, we can reuse the CSP for a given *one-shot learning of temporal action models* $\mathcal{L} = \langle F, I, G, A?, O, C\rangle$, to solve the following tasks:

- *Plan validation*. If $A?$ is a set of complete durative actions and given a plan $\pi = \{(a_1, t_{a_1}), (a_2, t_{a_2}) \dots (a_n, t_{a_n})\}$ then, the domain of all the CSP variables (X1,X2,X3,X4,X5,X6,X7,X8) is a singleton so plan validation reduces to checking whether this full assignment of the CSP is *consistent* with the input constraints. In the case of *inconsistency*, the plan can be executed starting from the initial state until an action condition is unsatisfied to identify the source of the plan failure.
- *Plan synthesis*. If $A?$ is a set of complete durative actions, then the value of the corresponding variables $X3$, $X4$, $X5$ is set as defined by $?A$ while the domains of the remaining variables is defined as explained in Section 4. In that case solving the CSP is equivalent to solving a temporal planning task $\mathcal{P} = \langle F, I, G, A?\rangle$ where the observations in $\mathcal{O}$ can be understood as a sequence of ordered *landmarks* [15] for $\mathcal{P}$ that are given as input (the fluents of the sets in $\mathcal{O}$ must be achieved by any plan that solves $\mathcal{P}$ and in the same order as defined in the observation $\mathcal{O}$).

What is more, our CSP formluation allows us to validate plans despite some of the variables that representing the conditions, effects and duration of an action do not have a fixed value and despite some (a, t_a) pairs are unknown or incomplete in the input plan to validate. In such scenario the plan validation ability of our CP formulation is beyond the functionality of VAL (the standard plan validation tool [16]) since it can address plan validation of partial, or even empty, action models and with partially observed plan traces (VAL requires both a full plan and a full action model for plan validation).

Likewise, we can synthesize a plan despite some of the variables that representing the conditions, effects and duration of an action do not have a fixed value (its value is initially unknown). This goes beyond the capacities of off-the-shelf planners that require a complete action model to synthesize a plan.

Last but not least, this integrative view of the tasks of *plan synthesis*, *plan validation* and the *learning* of action models applies, not only to the *temporal planning* setting but also to *classical planning*, the vanilla model of AI planning where actions are instantaneous and solutions are sequence of totally ordered actions [10]. In more detail, our CP formulation allows to transform the presented temporal planning model into a classical planning model by initially setting for every action $dur(a) = 0$ and adding the extra constraints $\mathsf{start}(a) = \mathsf{end}(a) = \mathsf{req\_start}(a) = \mathsf{req\_end}(a) = \mathsf{time}(f,a)$.

6 EVALUATION

[DE MOMENTO ESTO ESTA EN EL AIRE PORQUE NO SABEMOS COMO LO VAMOS A ABORDAR??]

The CP formulation has been implemented in Choco[3], an open-source Java library for constraint programming that provides an object-oriented API to state the constraints to be satisfied. Choco uses a static model of variables and constraints, i.e. it is not a DCSP.

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

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 Figure 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.

6.1 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 4.2.1. We first get the plans for these domains by using five planners (*LPG-Quality* [11], *LPG-Speed* [11], *TP* [18], *TFD* [6] and *TFLAP* [20]), 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 *struc*ture, the *dur*ation and the *struc*ture+*dur*ation, as discussed in section 5. 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 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.

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.

	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%

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.

6.2 Learning from scratch

7 CONCLUSIONS

REFERENCES

[1] Diego Aineto, Sergio Jiménez, Eva Onaindia, and Miquel Ramírez, 'Model recognition as planning', in *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 13–21, (2019).

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

[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] S. Edelkamp and J. Hoffmann, 'PDDL2.2: the language for the classical part of IPC–4', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS-04) – International Planning Competition*, pp. 2–6, (2004).

[6] 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).

[7] 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).

[8] 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).

[9] 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).

[10] 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).

[11] 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).

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

[13] Peter Gregory and Stephen Cresswell, 'Domain model acquisition in the presence of static relations in the lop system', in *Twenty-Fifth International Conference on Automated Planning and Scheduling*, (2015).

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

[15] Jörg Hoffmann, Julie Porteous, and Laura Sebastia, 'Ordered landmarks in planning', *Journal of Artificial Intelligence Research*, **22**, 215–278, (2004).

[16] 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).

[17] Yuxiao Hu, 'Temporally-expressive planning as constraint satisfaction problems.', in *ICAPS*, pp. 192–199, (2007).

[18] 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).

[19] 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).

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

[21] Sanjay Mittal and Brian Falkenhainer, 'Dynamic constraint satisfaction', in *Proceedings eighth national conference on artificial intelligence*, pp. 25–32, (1990).

[22] 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).

[23] Jussi Rintanen, 'Discretization of temporal models with application to planning with smt', in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, (2015).

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

[25] 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).

[26] Håkan LS Younes and Michael L Littman, 'Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects', *Techn. Rep. CMU-CS-04-162*, **2**, 99, (2004).

[27] 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).

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

[29] Hankz Hankui Zhuo, Tuan Nguyen, and Subbarao Kambhampati, 'Refining incomplete planning domain models through plan traces', in *Twenty-Third International Joint Conference on Artificial Intelligence*, (2013).

[30] 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).

[31] 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).