Learning Temporal Action Models via Constraint Programming

Antonio Garrido and **Sergio Jiménez**[1]

Abstract. We present a solver-independent Constraint Programming (CP) formulation for learning action models in temporal planning scenarios beyond PDDL2.1. Inspired by the CP approach for temporal planning, our formulation bases on a temporal plan trace and represents observations (as time-stamped states), actions, causal-link relationships, condition threats and effect interferences. This formulation is very expressive and supports a wide range of input knowledge. It also evidences the connection between the tasks of: i) action model learning, ii) plan validation, and iii) plan synthesis. Our experiments evaluate the quality of the learned models under different learning scenarios and in different planning domains.

1 INTRODUCTION

Temporal planning relaxes the assumption of instantaneous actions of *classical planning* [10]. Actions in temporal planning are *durative*, as they have a duration, and their conditions/effects may hold/happen at different times [7]. This means that *durative actions* can overlap in different ways [4]. Therefore, valid solutions for temporal planning instances must specify the precise time-stamp when durative actions start and end [15].

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 [17]. The more expressive the planning model, the more evident becomes this *knowledge acquisition bottleneck*, which 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 [22], we have seen systems able to learn action models with *quantifiers* [2, 28], from *noisy* actions or states [20, 24], from *null state information* [3], or from *incomplete* domain models [25, 27]. These systems use planning, SAT and genetic algorithm techniques, but they do not exploit constraint programming paradigms nor address the complex temporal planning aspects.

Most of the previous approaches are purely inductive and require large input datasets, e.g. hundreds of plan samples or observations, to compute statistically significant models. These approaches could learn a little from each sample, but not a complete valid model for a particular sample. We follow a different approach and address the learning setting where one (or more) model is learned from a single sample, i.e. one-shot learning. If hundreds of samples are available we will learn many models and return the most learned model, that is, the most repeated one.

[1] VRAIN, Valencian Research Institute for Artificial Intelligence. Universitat Politecnica de Valencia, Spain, email: {agarridot,serjice}@dsic.upv.es

As far as we know, this paper proposes the first approach for learning temporal action models. On the one hand, while learning action models 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 the time-stamps (temporal annotations) of conditions and effects and, when necessary, ii) estimate the actions duration. This contributes with an appealing way to learn from plan traces with overlapping actions in multi-agent environments [8]. On the other hand, our approach bases on the CP formulation of [9], which is used for planning and/or scheduling a whole plan. We keep the philosophy of using CP but, contrarily to [9], we address the inverse task now: learning the temporal action model given a plan trace. We also contribute with a solver-independent formulation that integrates the *learning* of temporal planning action models with their *synthesis* and *validation*.

2 BACKGROUND

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 a duration and conditions/effects on F at different times [9, 21]. To compactly represent temporal planning problems, we assume that the state variables in F are instantiations of a given set of predicates Ψ (like in the PDDL language [23]) and that durative actions in A are fully grounded from *operators*.

PDDL2.1 is the language for the temporal track of the International Planning Competition (IPC) [7, 12]. A PDDL2.1 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. $\mathsf{cond}_s(a), \mathsf{cond}_i(a), \mathsf{cond}_e(a)$ 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*), over the duration of a (*invariant/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.

Figure 1 shows an example of two PDDL2.1 durative actions from the *driverlog* domain of IPC. `board-truck` defines a fixed duration of two time units whereas 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 PDDL2.1 durative actions of the IPC-*driverlog* domain.

PDDL2.2 is an extension of PDDL2.1 that includes the notion of *Timed Initial Literal* (TIL) [13], denoted as $\mathsf{til}(p, t)$, and representing that variable $p \in F$ 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: $\mathsf{til}(\mathit{openWarehouse}, \mathit{8})$ and $\mathsf{til}(\neg \mathit{openWarehouse}, \mathit{20})$.

A *temporal plan* is a set $\pi = \{(a_1, t_{a_1}), (a_2, t_{a_2}) \dots (a_n, t_{a_n})\}$, where each pair (a_i, t_i) contains a durative action a_i and the start time $t_i = \mathsf{start}(a_i)$. The execution of π, 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 π. A *solution* to P is a plan π such that its execution, starting from s_0, is valid (i.e. it holds all the involved action conditions) and eventually satisfies $G \subseteq s_{end}$.

2.2 Constraint Satisfaction Problems

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 variables in X that is *consistent* with C. If we do not define a metric over X, many solutions, i.e. different variable assignments that are consistent with the input constraints, are possible and equally valid.

3 LEARNING A TEMPORAL ACTION MODEL

We formalize the task for *learning a temporal action model* 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 actions in $A?$ are *incomplete*. By incomplete we mean that the exact conditions+effects, their temporal annotation (*at start*, *over all* or *at end*), and the duration of actions are unknown. The operator, name and instantiated parameters (i.e. constants) of actions are known. The alphabet of $a \in A?$ (denoted $\alpha(a)$) is defined as the set of all predicates $p \in F$ that appear in the set of FOL interpretations over the instantiated parameters of a. For instance in the *driverlog* domain, $\alpha($`board-truck(driver1,truck1,loc1)`$) = \{$`(at driver1 loc1), (at truck1 loc1), (empty truck1), (driving driver1 truck1), (path loc1 loc1), (link loc1 loc1)`$\}$ of size 6. On the other hand, we formally define $candidates(a)$ as the two-set tuple $\langle\{p_i\}, \{p_i \cup \neg p_i\}\rangle$, where $p_i \in \alpha(a)$. The first set $\{p_i\}$ denotes all candidates that can be conditions of a, whereas the second set $\{p_i \cup \neg p_i\}$ denotes all candidates that can be effects of a. Without loss of generality, we learn positive conditions and positive+negative effects. Intuitively, $candidates(a)$ contains all the potential predicates that action a (or the corresponding operator) could learn; its size depends on the size of $\alpha(a)$, e.g. $|candidates($`board-truck(?d,?t,?l)`$)| = 6 * 3 = 18$, for any `?d`, `?t` and `?l`.
- O is the set of *observations* over a plan trace. It contains a full observation of I (time-stamped with $t = 0$) and a final state observation, which equals G (time-stamped with t_{end}, the *makespan* of the observed plan). Although I represents a full state observation, the final observation can represent a full or partial state: in plan synthesis and plan validation it is the partial goal state, and in learning it is the partial or full state to be explained by the learned model. O also contains the observations over the start and/or end times of actions and, optionally, other time-stamped observations of traversed intermediate partial states[2]. Figure 2 shows an example of O from the *driverlog* domain.
- C is an optional set of *constraints* that captures domain-specific expert knowledge. In this work these constraints are:
  - Mutually-exclusive (*mutex*) constraints that allow us to: i) automatically deduce new observations, and ii) prune action models inconsistent with these constraints. For instance, we can provide input knowledge to avoid drivers to be in two different locations at the same time. Hence, if we learn the driver is in one location we can automatically deduce an observation (s)he is no longer in the other locations. Figure 3 shows an example of six mutex constraints for the *driverlog* domain.
  - Constraints over $candidates(a)$ to represent partially specified action models [27]. For instance, we may know in advance that `path` and `link` are unnecessary for `board-truck`, while `path` is unnecessary for `drive-truck`. These constraints reduce the size of $candidates(a)$, thus improving the learning.

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 FORMULATING THE LEARNING TASK AS A CSP

Given a learning task $\mathcal{L}$ as defined in Section 3, we automatically create a solver-independent CSP whose solution induces an action model that solves $\mathcal{L}$.

[2] This work supports partial observations under the hypothesis that not all variables must be observed at any time. Observations are noiseless, which means that observed values are actual values with no uncertainty.

```
(: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 s2)
       (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))

(:goal (and (at driver1 s1) (at truck1 s1)))

(:observation :time 1
              :start (board-truck driver1 truck1 s2)
(:observation :time 11
              :start (drive-truck truck1 s2 s1 driver1)
...
(:observation :time 56 (at driver1 s1) (at truck1 s2))
(:observation :time 78 (at package1 s0) (at package2 s0))
```

Figure 2. Example of O containing a full state I, a partial state G, the start time of two actions, and two optional time-stamped partial states.

```
∀ ?d,?l1,?l2:    ¬at(?d,?l1) ∨¬at(?d,?l2), ≠(?l1,?l2)
∀ ?t,?l1,?l2:    ¬at(?t,?l1) ∨¬at(?t,?l2), ≠(?l1,?l2)
∀ ?t,?d:         ¬empty(?t) ∨¬driving(?d,?t)
∀ ?d,?t1,?t2:    ¬driving(?d,?t1) ∨¬driving(?d,?t2),
                 ≠(?t1,?t2)
∀ ?d1,?d2,?t:    ¬driving(?d1,?t) ∨¬driving(?d2,?t),
                 ≠(?d1,?d2)
∀ ?d,?l,?t:      ¬at(?d,?l) ∨¬driving(?d,?t)
```

Figure 3. Examples of six mutex constraints in C for the *driverlog* domain with drivers (?d, ?d1 and ?d2), locations (?l, ?l1 and ?l2) and trucks (?t).

4.1 The variables

For each action $a \in A?$ and predicate (condition or effect) $p \in candidates(a)$, we create the variables of Table 1. X1 represents the time-stamp when a starts, X2 when a ends and X3 its duration. The values 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 plan *makespan* (t_{end} if observed in O). If t_{end} is not observed, we consider a large enough domain for time. Boolean variables X4/X5 model whether p is actually a condition/effect of a. X6.1 and X6.2 define the interval throughout condition p must hold for the application of action a (provided $\mathsf{is\_cond}(p, a)$=*true*). X7 models a *causal link*, representing that action b supports p, which is required by a. If p is not a condition of a ($\mathsf{is\_cond}(p, a)$=*false*) then $\mathsf{sup}(p, a)$=$\emptyset$, representing an empty supporter. X8 models the time-stamp when effect p happens in a (provided $\mathsf{is\_eff}(p, a)$=*true*).

Table 1. The CSP variables, their domain and description.

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	$\mathsf{is\_cond}(p, a)$	$\{true, false\}$	true if p is a *condition* of a; false otherwise
X5	$\mathsf{is\_eff}(p, a)$	$\{true, false\}$	true if p is an *effect* of a; false otherwise
X6.1	$\mathsf{req\_start}(p, a)$,		
X6.2	$\mathsf{req\_end}(p, a)$	$[0..t_{end}]$	Interval when action a requires p
X7	$\mathsf{sup}(p, a)$	$\{b\}_{b \in A?} \cup \emptyset$	Supporters for causal link $\langle b, p, a\rangle$
X8	$\mathsf{time}(p, a)$	$[0..t_{end}]$	Time when the effect p of a happens

Additionally, we create two dummy actions:

- init, which represents the initial state I ($\mathsf{start}(\mathsf{init}) = 0$ and $\mathsf{dur}(\mathsf{init}) = 0$). It has no conditions so it has no associated variables is_cond, req_start, req_end and sup. It has as many $\mathsf{is\_eff}(p_i, \mathsf{init})$=*true* and $\mathsf{time}(p_i, \mathsf{init}) = 0$ as p_i in I.
- goal, which represents the goals G ($\mathsf{start}(\mathsf{goal}) = t_{end}$ and $\mathsf{dur}(\mathsf{goal}) = 0$). It has no effects so it has no is_eff and time variables. It has as many $\mathsf{is\_cond}(p_i, a)$=*true*, $\mathsf{sup}(p_i, \mathsf{goal}) \neq \emptyset$ and $\mathsf{req\_start}(p_i, \mathsf{goal}) = \mathsf{req\_end}(p_i, \mathsf{goal}) = t_{end}$ as p_i in G.

This formulation is powerful enough to model TILs and observations. A $\mathsf{til}(p, t)$ is analogous to init, and it is modeled as a dummy action that starts at time t and has instantaneous duration ($\mathsf{start}(\mathsf{til}(p, t)) = t$ and $\mathsf{dur}(\mathsf{til}(p, t)) = 0$) with no conditions and the single effect p that happens at time t ($\mathsf{is\_eff}(p, \mathsf{til}(p, t))$=*true* and $\mathsf{time}(p, \mathsf{til}(p, t)) = t$). An observation $\mathsf{obs}(p, t)$ is analogous to goal, and it is modeled as a dummy action that starts at time t and has instantaneous duration ($\mathsf{start}(\mathsf{obs}(p, t)) = t$ and $\mathsf{dur}(\mathsf{obs}(p, t)) = 0$) but with only one condition p, which is the value observed for p ($\mathsf{is\_cond}(p, \mathsf{obs}(p, t))$=*true*, $\mathsf{sup}(p, \mathsf{obs}(p, t)) \neq \emptyset$ and $\mathsf{req\_start}(p, \mathsf{obs}(p, t)) = \mathsf{req\_end}(p, \mathsf{obs}(p, t)) = t$), and no effects at all. Observations can also refer to $\mathsf{start}(a)$, $\mathsf{end}(a)$ or $\mathsf{dur}(a)$.

4.2 The constraints

Table 2 shows the constraints among the variables of Table 1. C1 and C2 model the end of any action, which must happen no later than goal. C3 models valid supporters. C4 forces to have a well-defined [req_start, req_end] interval, throughout condition p is required in a. C5 models that the time when b supports p must be before a requires it because of the causal link $\langle b, p, a\rangle$[3]. Given a causal link $\langle b, p, a\rangle$, C6 avoids the *threat* of action c deleting p (threats are solved via *promotion* or *demotion* [12]). C7 prevents action a from being a supporter of p when $\mathsf{is\_eff}(p, a)$=*false*. C8 models the fact that when the same action requires and deletes p 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 is checked instantly before the effect [7]. C9 prevents two actions from having contradictory effects at the same time. C10 only applies to non-dummy actions and forces them to have at least one condition and one effect (as usual, *true* is counted as 1 and *false* as 0).

Some conditions of Table 2 are redundant. For instance in C5 and C6, $\mathsf{sup}(p, a) = b$ means obligatorily $\mathsf{is\_eff}(p, b)$ = *true*. We include them here to define an homogeneous formulation but they are not included in our implementation. For simplicity, the value of some unnecessary variables is not bounded in the table. For instance, if $\mathsf{is\_cond}(p, a)$=*false*, the values for $\mathsf{req\_start}(p, a)$ and $\mathsf{req\_end}(p, a)$ become useless.

Specific constraints for PDDL2.1

Our formulation is more expressive than PDDL2.1. For instance, it allows conditions/effects to be at any time: constraint $\mathsf{req\_start}(p, a) = \mathsf{start}(a) - 2$ and $\mathsf{req\_end}(p, a) = \mathsf{start}(a) + 2$ easily allows condition p to hold in $\mathsf{start}(a) \pm 2$.

Making our formulation PDDL2.1-compliant is straightforward by adding the constraints of Table 3 for all non-dummy actions. C11

[3] $\mathsf{time}(p, b) < \mathsf{req\_start}(p, a)$ 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 p 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 their 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	**iff** ($\mathsf{is\_cond}(p, a)$=*false*) **then** $\mathsf{sup}(p, a) = \emptyset$	p is not a condition of $a \iff$ the supporter of p in a is $\emptyset$
C4	**if** ($\mathsf{is\_cond}(p, a)$=*true*) **then** $\mathsf{req\_start}(p, a) \leq \mathsf{req\_end}(p, a)$	$[\mathsf{req\_start}(p, a)..\mathsf{req\_end}(p, a)]$ is a valid interval
C5	**if** ($\mathsf{is\_eff}(p, b)$=*true*) **AND** ($\mathsf{is\_cond}(p, a)$=*true*) **AND** ($\mathsf{sup}(p, a) = b$)) **then** $\mathsf{time}(p, b) < \mathsf{req\_start}(p, a)$	Modeling the causal link $\langle b, p, a\rangle$: supporting p before it is required (obviously $b \neq \emptyset$)
C6	**if** ($\mathsf{is\_eff}(p, b)$=*true*) **AND** ($\mathsf{is\_cond}(p, a)$=*true*) **AND** ($\mathsf{is\_eff}(\neg p, c)$=*true*) **AND** ($\mathsf{sup}(p, a) = b$) **AND** ($c \neq a$) **then** ($\mathsf{time}(\neg p, c) < \mathsf{time}(p, b)$) **OR** ($\mathsf{time}(\neg p, c) > \mathsf{req\_end}(p, a)$)	Solving threat of c to causal link $\langle b, p, a\rangle$ by promotion or demotion (obviously $b \neq \emptyset$)
C7	**if** ($\mathsf{is\_eff}(p, a)$=*false*) **then forall** b that requires p: $\mathsf{sup}(p, b) \neq a$	a cannot be a supporter of p for any action b
C8	**if** ($\mathsf{is\_cond}(p, a)$=*true*) **AND** ($\mathsf{is\_eff}(\neg p, a)$=*true*) **then** $\mathsf{time}(\neg p, a) \geq \mathsf{req\_end}(p, a)$	a requires and deletes p: the condition holds before the effect
C9	**if** ($\mathsf{is\_eff}(p, b)$=*true*) **AND** ($\mathsf{is\_eff}(\neg p, c)$=*true*) **then** $\mathsf{time}(p, b) \neq \mathsf{time}(\neg p, c)$	Solving effect interference at the same time (p and $\neg p$)
C10	**forall** condition p_i and effect p_j of a: $\sum \mathsf{is\_cond}(p_i, a) \geq 1$ **AND** $\sum \mathsf{is\_eff}(p_j, a) \geq 1$	Every non-dummy action has at least one condition and effect

limits the conditions to be only *at start*, *over all* or *at end*. C12 limits the effects to happen *at start* or *at end*. In PDDL2.1, all actions $\{a_j\}$ grounded from the same operator share the same structure of conditions/effects. C13 guarantees this for the conditions and C14 for the effects. C15 makes the duration of all occurrences, which can happen many times and are modeled separately, of the same action equal (this is optional in PDDL2.1). C16 forces all actions to have at least one of its *n*-effects *at end*, since actions with only *at start* effects turn the value of the duration irrelevant. Strictly speaking, C16 is not specific of PDDL2.1, but it requires to have *at end* effects and produces more rationale models for their durative actions.

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

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

Mutex constraints

As seen in Section 3, mutex constraints in C can be exploited as input knowledge to automatically deduce new observations in $\mathcal{L}$. If two predicates $\langle p_i, p_j\rangle$ are mutex they cannot hold simultaneously, and the learned action model needs to satisfy this. Consequently, if p_i holds, we can infer $\neg p_j$ (despite $\neg p_j$ was not actually observed). This reasoning is specially relevant for correctly learning *negative effects* when there is a lack of input observations. After all, what is the necessity to learn negative effects if they are not required nor directly observed? Mutex reasoning helps us to fill this void by automatically inferring the observation of negated variables, which forces later to satisfy the *causal links* of negative variables, and improves the learned models (as we will see in Section 6). Note however that, given a $\langle p_i, p_j\rangle$-mutex in a durative actions setting, $\neg p_i$ does not necessarily imply p_j. See, for example, the effects `(not (at ?t ?l1))` and `(at ?t ?l2)` of action `drive-truck` in Figure 1, which respectively happen at different times (*at start vs. at end*). As defined in Figure 3, these two predicates are mutex as a truck cannot be in two locations simultaneously; although it is valid for a truck to be, for some time (from start to end), at no location. Note this situation does not happen in classical planning, where actions have instantaneous effects and if $\langle p_i, p_j\rangle$ are mutex, then p_i implies $\neg p_j$ and vice versa.

Dynamic observations are necessary to exploit mutex constraints at any intermediate state, even if such state was not observed at all. Reasoning on a mutex $\langle p_i, p_j\rangle$ means that, immediately after a asserts p_i we need to ensure the observation $\neg p_j$. Technically, when $\mathsf{is\_eff}(p_i, a)$ takes the value *true*, then the observation $\mathsf{obs}(\neg p_j, \mathsf{time}(p_i, a) + \epsilon)$ needs to be dynamically added. The time of the observation cannot be just $\mathsf{time}(p_i, a)$, as we first need to assert p_i and one ϵ later observe $\neg p_j$. Adding the variables and constraints for this new observation during the CSP search is trivial for *Dynamic CSPs* (DCSPs) [19]. Otherwise, we need to statically define a new type of observation $\mathsf{obs}(p_i, a, \neg p_j)$, where a supports p_i which is mutex with p_j and, consequently, we will need to observe $\neg p_j$. The difference *w.r.t.* an original obs is twofold: i) the observation time is now initially unknown, and ii) the observation will be activated or not according to the following constraint:

if ($\mathsf{is\_eff}(p_i, a)$=*true*) **then** ($\mathsf{start}(\mathsf{obs}(p_i, a, \neg p_j)) = \mathsf{time}(p_i, a) + \epsilon$) **AND** ($\mathsf{is\_cond}(\neg p_j, \mathsf{obs}(p_i, a, \neg p_j))$=*true*)
else $\mathsf{is\_cond}(\neg p_j, \mathsf{obs}(p_i, a, \neg p_j))$=*false*

Reasoning on mutex depends on optional input knowledge in C and increases notably the size of our formulation, specially in non-DCSPs, but it is automated together with the creation of all the constraints of Table 2.

4.3 The heuristics

In a pure satisfaction problem, all possible solutions are equally valid. We have investigated the use of several metrics (e.g. reducing the number of causal links or side effects), and although they allow the user to specify preferences over the space of possible solutions, we have not found a metric that leads to learn the best model. Therefore, we have focused on simple heuristics that show effective in the tradeoff quality of learning *vs.* performance, and guide the search in a univocal way. Hence, we propose the following variable+value ordering heuristics:

1. X4 ($\mathsf{is\_cond}$). True first, which learns the most restrictive model of conditions.
2. X5 ($\mathsf{is\_eff}$). False first, which learns a model with the min number of causal links, which reduces the number of side effects.
3. X8 (time). Lower values first for negative effects, while upper values first for positive effects. This learns delete and positive effects as eff_s and eff_e, respectively.
4. X6 ($\mathsf{req\_start}$ and $\mathsf{req\_end}$). Lower values first for $\mathsf{req\_start}$, while upper values first for $\mathsf{req\_end}$. This gives priority to cond_i, trying to keep conditions as long as possible in the model learned.
5. X7 (sup). Lower values first to learn supporters that start earlier in the plan trace.

6. X3 (dur). Lower values first, which learns a model with the shortest actions.

5 A UNIFIED FORMULATION FOR PLANNING, VALIDATION AND LEARNING

Our formulation has been primarily designed to solve the task of *model learning*, but it is strongly connected to the tasks of plan *synthesis* and plan *validation*. This connection lies on the fact that we can leverage the input knowledge on the planning domain over a wide range of (un)known levels. This section provides an integrative view for these three tasks in a temporal planning setting, although this connection also applies to a classical planning model[4], that is, the vanilla model of planning where actions are instantaneous and a solution is a totally ordered sequence of actions [10].

Plan synthesis Given a learning task $\mathcal{L} = \langle F, I, G, A?, O, C\rangle$, each action in $A?$ is completely specified. This is equivalent to know in advance the values for variables {X3,X4,X5} of Table 1 and the OR-constraint that holds in {C13,C14} of Table 3, while other variables/constraints remain open/unknown. The observations in O are incomplete as no information on the start/end time of actions is given, i.e. the values of {X1,X2} are to be determined. This provides the complete model of actions, with the duration, conditions and effects (and their temporal annotation), as defined in the PDDL2.1 domain. In this case, solving the resulting CSP is equivalent to solve the temporal planning problem $P = \langle F, I, G, A?\rangle$. The solution is a plan that reaches G from state I under the complete action model defined in $A?$. Also, the observations over a plan trace in O can be understood as a sequence of time-stamped *landmarks* [14] for P that are given as input (the predicates of the sets in O must be achieved by any plan that solves P and at the time-stamps given by O).

Moreover, it is important to note that our formulation allows us to synthesize a plan despite some of the variables that represent the conditions, effects and duration of an action are unknown. This subsumes the capabilities of off-the-shelf planners that require the complete model of actions for planning.

Plan validation Given a learning task $\mathcal{L} = \langle F, I, G, A?, O, C\rangle$, each action in $A?$ is completely specified like in plan synthesis. The observations over a plan trace in O are specified like in plan synthesis and, additionally, the observations on the start/end times of actions are also complete, which means that the values of {X1,X2} are now known. This provides both the complete model of actions, as defined in the planning domain, and the complete plan trace for all actions (the temporal plan $\pi = \{(a_1, t_{a_1}), (a_2, t_{a_2}) \ldots (a_n, t_{a_n})\}$ is consequently known). This allows us to know in advance the values of all variables {X1,X2… X8}. In this case, solving the resulting CSP is equivalent to check whether this full assignment of the CSP is consistent, which means validating the plan π. In the event of *inconsistency*, π will need to be executed from I until a condition is unsatisfied to identify the source of the plan failure.

Moreover, our formulation allows us to validate plans despite some of the variables that represent the conditions, effects and duration of an action are unknown, and despite some (a, t_a) pairs in the input plan trace are incomplete. In such scenarios, the plan validation ability of our formulation is beyond the functionality of VAL (the standard plan validation tool [15]) since it can address plan validation of partial, or even empty, action models and with partially observed plan traces. On the contrary, VAL requires both a full plan and a full action model for plan validation. Note, however, that finding a solution for our formulation means it is verified at least once, rather than no matter what will be filled in there, like VAL checks.

[4] In practice, we can transform our temporal planning model into a classical one by setting for every action $\mathsf{dur}(a) = 1$ and adding the extra constraint $\mathsf{start}(a) = \mathsf{end}(a) = \mathsf{req\_start}(?, a) = \mathsf{req\_end}(?, a) = \mathsf{time}(?, a)$.

Action model learning Given a learning task $\mathcal{L} = \langle F, I, G, A?, O, C\rangle$, we can learn the action model from scratch by using our formulation, which is the most expensive and less scalable task; it requires finding a solution to the resulting CSP that builds the full action model. But we can also learn from several partially specified action models, thus simplifying the learning task. Solving the resulting CSP in those cases is equivalent to *fill the gaps* of a partial action model where, optionally:

- Some conditions and/or effects are known (some {X4,X5} have initial values); e.g. we know that an action requires for sure certain conditions and we are mainly interested in learning their temporal annotations (*at start*, *over all* or *at end*).
- The temporal annotation of some conditions and/or effects is known (we know some OR-constraints that hold in {C13,C14}); e.g. we know that some effects always happen *at end* of the action.
- Some start/end times or durations are known (some {X1,X2,X3} have initial values); e.g. the duration of some actions is known.

In addition to the three previous options that simplify and improve the performance of the learning task, there are two additional options that can help us to improve the quality of the learned models where:

- Some $candidates(a)$ are filtered in a. If the alphabet $\alpha(a)$ is big, the size of $candidates(a)$ can be huge. This not only makes the learning task more expensive but also facilitates a bad learning of predicates. A *static predicate* represents information that is always true and never changes. For instance, as shown in Section 3, {`(path loc1 loc2)`, `(link loc1 loc2)`} represent the fact that there is a path/link between `loc1` and `loc2`. They can be necessary in the conditions of some actions (e.g. `drive-truck`), but never in the effects so they should never be learned as effects. Therefore, as a preprocessing stage to improve the input knowledge of $A?$, we can filter static predicates from the effects-set of $candidates(a)$ and make it smaller.
- The final state observation in G contains a full goal state, i.e. the total assignment of the last state variables, rather than a partial assignment. Similarly to the mutex reasoning, having a full state forces to satisfy the causal links of all state variables, which reduces performance but improves the quality of the learning. Intuitively, since G is more informative now, the learning is better.

Finally, it is important to note that our learning task learns one model from one plan trace. A more general learning task could learn one model from dozens of plan traces but, though very appealing, this has serious problems of scalability (the number of variables+constraints grows alarmly). To learn from many traces, we can apply the learning task to each individual trace and then return the most learned model, i.e. the most repeated one. The benefit here is twofold: i) the overall learning time is shorter; and ii) the learned model explains the 100% of, at least, one plan trace. Although this seems too obvious, it is the main limitation of the learning approaches that learn statistically models from many samples.

6 EVALUATION

Our formulation has been implemented in Choco (`http://www.choco-solver.org`), an open-source Java library for CP that provides an object-oriented API. Choco uses a static model of variables and constraints, i.e. it is not a DCSP. We have used this implementation to evaluate the quality of the learned models, from both a syntactic and semantic perspective. We have used four IPC PDDL2.1 domains, thus needing their specific constraints of Section 4.2, and solved a collection of instances by using five planners (*LPG-Quality* [11], *LPG-Speed* [11], *TP* [16], *TFD* [6] and *TFLAP* [18]). We have randomly chosen 50 plans (up to 20-30 actions) per domain, which are automatically compiled into 50 learning tasks that configure our experiments dataset. Then we solved each learning task, in satisfaction mode, and got the first five action models found. We also calculate the most learned models. The solving time was limited to 300s on an Intel i5-6400 @ 2.70GHz with 8GB of RAM.

There are several elements in a learning task $\mathcal{L}$ that can be considered, as seen in Sections 4.2 and 5. First, we can enable mutex reasoning or not (named **mutexON** or **mutexOFF**). Second, the static effect predicates can be filtered in $candidates$ of all operators as a preprocessing stage (denoted as **+F**) or not. Third, the final state in G means observing Only partial Goals (named **OG**) or observing the Full goal State (named **FS**). This way, we run six learning scenarios: OG, OG+F and FS+F, each with mutexON and mutexOFF (we discard FS with no Filtering because of the size of G).

6.1 Experimental setup

Table 4 summarizes our experiments. #O is the number of Operators and #PTL the number of Predicates To Learn. #candidates is the size of $candidates$ for all the operators when no Filtering (not +F) and when such Filtering (+F) is done. The number of #tasks solved is given for the six learning scenarios used in this section: OG, OG+F, FS+F, and these in both mutexOFF (first line) and mutexON (second line) versions. For instance, in *zenotravel* we need to learn 5 operators and 28 predicates. In absence of filtering, the number of candidates is 105, which is reduced to 71 when +F. This reduction depends on the domain definition, and ranges from zero (*parking*) to significant values (*floortile*). As can be seen, not all the tasks were solved in 300s. This depends on the complexity of the domain, the #PTL and, specially, the #candidates. Dealing with a scenario task of OG is easier because there are fewer predicates in G (though this is less informative) than in the FS version. Furthermore, learning with mutexOFF is usually easier than learning with mutexON because no deduced observations are necessary, but less informative.

Table 4. Summary of our experiments. For the #tasks solved, the first line reports the results for mutexOFF and the second line for mutexON.

	#O	#PTL	#candidates		#tasks solved		
			not +F	+F	**OG**	**OG+F**	**FS+F**
zenotravel	5	28	105	71	50 (**0.27**) 50 (**0.26**)	50 (**0.11**) 50 (**0.21**)	50 (**0.04**) 50 (**0.02**)
driverlog	6	28	144	96	49 (**0.34**) 42 (**0.39**)	50 (**0.24**) 50 (**0.33**)	42 (**0.28**) 46 (**0.31**)
floortile	7	44	417	217	50 (**0.62**) 28 (**0.89**)	50 (**0.48**) 49 (**0.97**)	21 (**0.48**) 21 (**0.48**)
parking	4	32	131	131	50 (**0.67**) 50 (**0.82**)	50 (**0.60**) 50 (**0.51**)	50 (**0.36**) 50 (**0.45**)

In *zenotravel* the 50 tasks were solved for all six scenarios. Since five action models are returned per solved task, up to 250 potential different models are to be learned per scenario in *zenotravel*. However, this is not the case and this number is usually lower, which is a good indication that models *tend to converge* more easily; i.e. many different tasks learn the same model. This indication of convergence is depicted in the table between brackets and in bold text, as the relation between the number of different learned models and the potential number of models (clearly lower values are better). In *zenotravel* OG+mutexOFF only 67 different models were found, which gives a value of 67/250=0.27. These values are good in *zenotravel* (particularly in OG+F and FS+F) and *driverlog*, and worse in *parking* and *floortile*, where OG+F+mutexON shows the worst result (0.97).

6.2 Syntactic evaluation. Precision and recall

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. Hence, the aim is to assess the precision and recall of the learned model, two common metrics in learning [1, 26, 28], that give us an intuitive idea on the soundness and completeness, respectively, of the new model.

Given two models, $precision = \frac{p^{=}}{p^{=}+p^{\neg}}$, where $p^{=}$ counts the number of predicates (i.e. conditions+effects) that appear correctly and are temporally annotated equally in both models, and $p^{\neg}$ counts the number of predicates that appear in the learned model but should not appear. On the other hand, $recall = \frac{p^{=}}{p^{=}+p^{\neq}}$, where $p^{\neq}$ counts the number of predicates that should appear in the learned model but are not present. Table 5 depicts these metrics for our six learning scenarios as average scores for all the learned models. We show the scores for the Start, Invariant and End Conditions (SC, IC and EC respectively), Start and End Effects (SE and EE respectively), and All Conditions and All Effects (AC and AE respectively) for the six learning scenarios (mutexOFF and mutexON are shown in the first and second line, respectively).

The use of mutexON has a positive impact in the precision of AC, but not in AE, and improves the scores of the recall in both AC and AE. Note the recall of SE, which is 0 for mutexOFF in OG and OG+F, and significantly higher when mutexON. With mutexOFF there is no need to learn negative effects, typically modeled as start effects, and the learned models are *relaxed* models where negative effects are not included. FS generally improves the precision of AC and AE, and also improves the recall of AE because the negative effects present in the full final state cannot be relaxed so need to be learned. Consequently, mutexON or FS help to improve the completeness of the learned effects. The Filtering scenarios (+F) improves the precision of AC and AE, specially where there exists irrelevant static information (e.g. *floortile*). Due to lack of space in the table, we do not show all the precision and recall scores for the most learned model. But we do show its precision and recall of AC and AE (between brackets and in bold text). Although the most learned model generally produces scores above the average, this cannot be guaranteed. Actually, we have detected that sometimes there are several most learned models, i.e. different models that are learned the same number of times. But we have not found a safe tie-breaking mechanism to decide the model that leads to the best scores.

6.3 Semantic evaluation. Validation

There is not a unique reference model when learning temporal models; e.g. *at start* and *over all* can be interchangeable in some domains, but they are syntactically different. Consequently, a pure syntax-based measure might return misleading results. From this standpoint,

Table 5. Precision and recall scores.

OG	Precision; mutexOFF: first line, mutexON: second line							Recall; mutexOFF: first line, mutexON: second line						
	SC	IC	EC	SE	EE	AC	AE	SC	IC	EC	SE	EE	AC	AE
zenotravel	0.62	0.23	0.94	0.80	0.81	0.60 **(0.76)**	0.81 **(1.0)**	0.09	0.92	1.0	0.0	0.72	0.64 **(0.72)**	0.36 **(0.40)**
	0.88	0.43	0.84	0.66	0.76	0.72 **(0.83)**	0.71 **(0.88)**	0.75	0.69	1.0	0.97	0.70	0.81 **(0.89)**	0.84 **(0.90)**
driverlog	0.25	0.17	0.86	0.59	0.52	0.43 **(0.73)**	0.56 **(0.72)**	0.04	0.95	1.0	0.0	0.93	0.65 **(0.73)**	0.47 **(0.72)**
	0.60	0.20	0.74	0.66	0.28	0.51 **(0.61)**	0.47 **(0.37)**	0.71	0.60	1.0	0.92	0.70	0.77 **(0.81)**	0.81 **(0.65)**
floortile	0.17	0.15	0.44	0.44	0.25	0.25 **(0.38)**	0.35 **(0.66)**	0.02	0.96	1.0	0.0	0.68	0.65 **(0.67)**	0.34 **(0.34)**
	0.73	0.27	0.36	0.61	0.27	0.45 **(0.36)**	0.44 **(0.62)**	0.76	0.90	1.0	0.80	0.77	0.89 **(0.92)**	0.79 **(1.0)**
parking	0.62	0.0	0.76	0.98	0.76	0.46 **(0.33)**	0.87 **(0.85)**	0.05	1.0	1.0	0.0	0.94	0.67 **(1.0)**	0.47 **(1.0)**
	0.76	0.96	0.50	0.50	0.48	0.74 **(0.92)**	0.49 **(0.54)**	0.82	1.0	1.0	0.92	0.76	0.94 **(0.67)**	0.84 **(0.50)**

OG+F	Precision; mutexOFF: first line, mutexON: second line							Recall; mutexOFF: first line, mutexON: second line						
	SC	IC	EC	SE	EE	AC	AE	SC	IC	EC	SE	EE	AC	AE
zenotravel	0.52	0.25	0.98	0.82	1.0	0.58 **(0.75)**	0.91 **(1.0)**	0.08	0.87	1.0	0.0	0.73	0.62 **(0.73)**	0.37 **(0.35)**
	0.82	0.38	0.74	0.65	0.89	0.65 **(0.53)**	0.77 **(0.88)**	0.67	0.47	1.0	0.95	0.68	0.71 **(0.50)**	0.82 **(0.80)**
driverlog	0.33	0.23	0.96	0.64	0.90	0.51 **(0.74)**	0.77 **(0.61)**	0.06	0.94	1.0	0.0	0.93	0.65 **(0.67)**	0.47 **(0.50)**
	0.58	0.34	0.87	0.64	0.43	0.60 **(0.52)**	0.54 **(0.54)**	0.68	0.49	1.0	0.87	0.73	0.72 **(0.56)**	0.80 **(0.84)**
floortile	0.39	0.23	0.74	0.56	0.77	0.45 **(0.75)**	0.67 **(0.80)**	0.06	0.99	1.0	0.0	0.70	0.66 **(0.67)**	0.35 **(0.38)**
	0.72	0.41	0.51	0.60	0.47	0.55 **(0.79)**	0.54 **(0.42)**	0.71	0.93	1.0	0.72	0.66	0.88 **(0.79)**	0.69 **(0.36)**
parking	0.62	0.0	0.78	0.98	0.76	0.47 **(0.67)**	0.87 **(0.85)**	0.05	1.0	1.0	0.0	0.94	0.67 **(0.67)**	0.47 **(0.50)**
	0.76	0.99	0.48	0.50	0.48	0.74 **(0.92)**	0.49 **(0.47)**	0.80	1.0	1.0	0.90	0.75	0.93 **(1.0)**	0.83 **(1.0)**

FS+F	Precision; mutexOFF: first line, mutexON: second line							Recall; mutexOFF: first line, mutexON: second line						
	SC	IC	EC	SE	EE	AC	AE	SC	IC	EC	SE	EE	AC	AE
zenotravel	0.84	0.42	0.90	0.71	0.99	0.72 **(0.83)**	0.85 **(0.88)**	0.70	0.60	1.0	1.0	0.75	0.53 **(0.89)**	0.88 **(0.90)**
	0.85	0.51	0.88	0.65	0.99	0.75 **(0.83)**	0.82 **(0.88)**	0.71	0.59	1.0	0.99	0.74	0.77 **(0.89)**	0.87 **(0.90)**
driverlog	0.60	0.36	0.98	0.64	0.94	0.65 **(0.67)**	0.79 **(0.84)**	0.57	0.89	1.0	0.64	0.95	0.63 **(0.83)**	0.80 **(0.84)**
	0.62	0.34	0.69	0.69	0.57	0.55 **(0.44)**	0.63 **(0.61)**	0.74	0.48	1.0	0.96	0.83	0.74 **(0.86)**	0.90 **(0.88)**
floortile	0.69	0.35	0.62	0.66	0.74	0.55 **(0.37)**	0.70 **(0.84)**	0.67	0.96	1.0	0.66	0.73	0.65 **(0.93)**	0.70 **(1.0)**
	0.70	0.45	0.39	0.59	0.67	0.51 **(0.38)**	0.63 **(0.65)**	0.86	0.88	1.0	0.89	0.87	0.91 **(0.92)**	0.88 **(1.0)**
parking	0.86	0.0	1.0	0.89	0.89	0.62 **(0.67)**	0.89 **(1.0)**	0.53	1.0	1.0	0.72	0.95	0.67 **(0.87)**	0.84 **(0.92)**
	0.74	1.0	0.50	0.50	0.46	0.75 **(0.58)**	0.48 **(0.46)**	0.87	1.0	1.0	0.94	0.82	0.96 **(0.93)**	0.88 **(0.85)**

the quality of the learned model can be assessed by analyzing the success ratio of the learned model *against* unseen samples of a test dataset, analogously to a classification task.

Formally, $success\ ratio = \frac{samples^+}{|dataset|}$, where $samples^+$ counts the number of samples the learned model explains on a test $dataset$. A ratio of 1.0 implies learning a model that explains the full dataset: a solution is found which is consistent with the constraints of the learned model together with the test samples ones.

Table 6 shows the average success ratios, where each learned model is validated against the remaining tasks of the same domain. For instance, in *zenotravel* there are 50 tasks solved per learning scenario (see Table 4). The five models of each task should be validated against the 49 remaining tasks, which means a huge evaluation (250*49=12250 instances). For simplicity, we only consider the first model learned per task, and the experiment contains 50*49=2450 evaluations per scenario. As usual, mutexOFF and mutexON are shown in the first and second line, respectively The best results are in *zenotravel* and the worst results in *floortile*, which corresponds with the *convergence tendencies* shown in Table 4. The semantic evaluation for the most learned model against the remaining tasks is shown between brackets and in bold text. Such a model generally produces ratios above the average in most scenarios, but this cannot be guaranteed for all the domains.

Table 6. Success ratio of the models learned *against* the test dataset.

	OG	**OG+F**	**FS+F**
zenotravel	0.77 **(1.0)**	0.88 **(0.80)**	0.95 **(1.0)**
	0.77 **(1.0)**	0.91 **(1.0)**	1.0 **(1.0)**
driverlog	0.40 **(0.63)**	0.69 **(0.71)**	0.52 **(0.51)**
	0.44 **(0.61)**	0.67 **(0.63)**	0.56 **(0.69)**
floortile	0.22 **(0.04)**	0.29 **(0.67)**	0.32 **(0.60)**
	0.34 **(0.41)**	0.31 **(0.21)**	0.31 **(0.35)**
parking	0.40 **(0.37)**	0.40 **(0.37)**	0.71 **(0.92)**
	0.52 **(0.59)**	0.48 **(1.0)**	0.65 **(1.0)**

All in all, we have found out the semantic evaluation is a bit convoluted. One subtle syntactic difference might not affect the semantic evaluation (e.g. interchangeable conditions). On the contrary, an effect that is not correctly learned involves a subtle penalization in the syntactic evaluation, but it affects negatively the semantic evaluation (that difference might not explain a huge number of samples, as usually happens in *floortile*). Therefore, we have detected that in some domains the success ratio can also return misleading results.

7 CONCLUSIONS

There is a growing interest for learning action models in AI planning due to its application to recognition of past behavior for prediction, decision taking, robotics motion capturing, etc. Learning is appealing because these scenarios include a huge number of tasks, sometimes difficult to be described formally, which require expert knowledge that is impractical in complex domains.

The general contribution of this paper is a solver-independent CP formulation to learn action models in temporal planning, which is more complex than in classical planning because actions can overlap in different ways. We have formulated all variables and constraints under a flexible schema that accommodates a high level of expressiveness, where all relations of Allen's algebra for temporal reasoning are supported. What is more, we also support different levels of specification of the input knowledge. This knowledge, as a partially specified action model, can be adapted to address not only the learning task but also the planning and validation tasks.

We have proposed variable+value ordering heuristics that prove effective in our experiments, which test different learning scenarios. As a summary, the results show that reasoning on mutex (mutexON) is more expensive than mutexOFF, but it improves the quality of the learned models. Filtering static predicates reduces the number of decisions to take, simplifies the task and improves the learning. Observing only partial goal states (OG) is easier than observing full goal states (FS), but the latter provides a more complete learning. In general, mutexON or FS help to learn negative effects, which in other scenarios are relaxed and not learned. FS is useful when (negative) effects do not change frequently throughout the plan trace, whereas mutexON is very useful to learn strong interactions between contradictory predicates (when one is true the other should be immediately and automatically observed as false).

Our formulation can be solved by Satisfiability Modulo Theories, which is part of our current work. As for future work, we want to extend our formulation to learn from intermediate observations (we need to investigate how many and how frequent they must be) and to learn meta-models (as combinations of several learned models).

ACKNOWLEDGEMENTS

This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R. S. Jimenez is partly supported by the RYC15/18009.

REFERENCES

[1] D. Aineto, S. Jiménez, and E. Onaindia, 'Learning STRIPS action models with classical planning', in *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS-18)*, pp. 399–407, (2018).

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

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

[15] 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).

[16] 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).

[17] 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).

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

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

[20] 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).

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

[22] 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).

[23] 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).

[24] 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).

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

[26] Hankz Hankui Zhuo, Hector Muñoz Avila, and Qiang Yang, 'Learning hierarchical task network domains from partially observed plan traces', *Artificial Intelligence*, **212**, 134–157, (2014).

[27] 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).

[28] 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).