One-Shot Learning of Concurrent Durative [o Temporal??] Actions Models [via Constraint Programming???]

Antonio Garrido and **Sergio Jiménez**

Abstract. We present a *Constraint Programming* (CP) formulation for learning models of *durative actions* from the observation of a single plan execution. Inspired by the CSP approach to *temporal planning*, our CP formulation models *time-stamps* for actions, *causal-link* relationships, *threats* and effect *interferences* and evidences the connection between the tasks of *plan synthesis*, *plan validation* and *action model learning*. The CP formulation is solver-independent so off-the-shelf CSP solvers can be used for the resolution of any of these tasks. The performance of our CP formulation is evaluated learning and validating action models of several temporal domains specified in PDDL2.1. The paper also shows that the CP formulation is flexible to accommodate a different range of expressiveness, subsuming the PDDL2.1 temporal semantics.

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*, because each action has an associated duration and hence, the conditions/effects of an action 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 must indicate the precise time-stamp when durative actions start and end [14].

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 (temporal) planning models [16]. The more expressive the planning model, the more evident becomes this knowledge acquisition bottleneck that jeopardizes the usability of AI planning technology. There is however a growing interest in the planning community for the machine learning of action models [17, 19, 21, 22] with a wide range of different approaches for learning classical action models from sequential plans [2]. Since pioneering learning systems like ARMS [21], we have seen systems able to learn action models with quantifiers [1, 25], from noisy actions or states [19, 22], from null state information [3], or from incomplete domain models [23, 24].

As far as we know this work is the first approach for learning action models for temporal planning. 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 action, and ii) estimate the action duration. Further, 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 and focus on learning models from sequential plans for classical planning.

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. The contributions of this work are two-fold:

1. We show how to learn action models from observations of plans with overlapping actions. This feature makes our approach appealing for learning action models in multi-agent environments [8].
2. A CP formulation that connnects the *learning* of planning action models with the *synthesis* and the *validation* of plans. The paper shows that of-the-shelf CSP solvers can be used for any of these tasks. Further, we show that the plan validation ability of our CP formulation is beyond the functionality of VAL (the standard plan validation tool [14]) 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).

2 BACKGROUND

This section formalizes the *temporal planning* and *Constraint Satisfaction* models 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. $|I| = |F|$) that is time-stamped with $t = 0$; $G \subseteq F$ is a conjunction of *goal conditions* over the variables in F that defines the set of goal states; and A represents the set of *durative actions*. A *durative action* has an associated duration and may have conditions/effects on F at different times [9, 20]. In this work we assume that durative actions in A are fully grounded from *action schemes* (aka *operators*) to compactly represent temporal planning problems.

PDDL2.1 is the input 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 since the semantics of a PDDL2.1 durative action a can always be defined in terms of just two discrete events, $\mathsf{start}(a)$ and $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. This means that if 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).

A *temporal plan* is a set of pairs $\pi = \{(a_1, t_1), (a_2, t_2) \dots (a_n, t_n)\}$. Each pair (a_i, t_i) contains a durative action a_i and a time-stamp $t_i = \mathsf{start}(a_i)$. The execution of a temporal plan starting from a given initial state I induces a state sequence formed by the union of all states $\{s_{t_i}, s_{t_i+\mathsf{dur}(a_i)}\}$, where there exists an initial state $s_0 = I$, and a state s_{end} that is the last state induced by the execution of the plan. Note then that sequential plans can 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, $G \subseteq s_{end}$.

PDDL2.2 is an extension of the language PDDL2.1 that includes the notion of *Timed Initial Literal* [13], denoted as $\mathsf{til}(f, t)$, and representing that a variable $f \in F$ becomes true (or false) at a certain time $t > 0$, independently of the actions in the plan [5]. Traditionally, TILs are useful to model *exogenous happenings*; for instance, a time window when a warehouse is open in a logistics scenario, $\mathsf{til}(open, 8)$ and $\mathsf{til}(\textit{not-open}, 20)$.

2.2 Constraint Satisfaction

A *Constraint Satisfaction Problem* (CSP) is a tuple $\langle X, D, C\rangle$, where X is a set of finite variables, D represents the finite 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 as an assignment of values to all variables in X that satisfy all the constraints in C, that is, those values are *consistent* with all the constraints.

Given a CSP there may be many different solutions to that problem. A *cost-function* can be defined to specify user preferences about the space of possible solutions. Given a CSP and cost-function, then an *optimal solution* is a total assignment of the variables that is consistent with the constraints of the CSP and minimizes the value of the input cost-function.

3 LEARNING TEMPORAL ACTION MODELS

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

3.1 The hypothesis space

[SUGERENCIA A VER QUE TE PARECE?? 1. LOS DOS PARRAFOS DE VOCABULARY. 2. SE INDICA QUE ES LO QUE TENEMOS COMO HIPOTESIS DE PARTIDA DONDE UNA ACCION TIENE UN CONJUNTO CANDIDATO (HABRIA QUE RECALCAR ESTO DE **CANDIDATE**) DE CONDS/EFFS (COMO LO QUE DEVUELVE EL COMPILER). 3. LO QUE SE DEBE CONSEGUIR QUE ES EL EJEMPLO DE LAS DOS ACCIONES. OTRA ALTERNATIVA SERIA HACER 1-3-2. AUNQUE NO TENGO CLARO SI LO DE CANDIDATE DEBERIA IR A 3.1]

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

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

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

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

Like in PDDL, we assume that the set F of state variables is given by the instantiation of a given set of predicates Ψ. We denote as $\mathcal{I}_{\xi,\Psi}$ the *vocabulary* (set of symbols) that can appear in the conditions and effects of a given durative action schema ξ. This set is formally defined as the FOL interpretations of predicates Ψ, over the action parameters $pars(\xi)$. [NO SE PUEDE SIMPLIFCAR ESTO DE ALGUNA FORMA??? NO HEMOS DICHO QUE HAYA PARAMETROS!! NO LO PODEMOS PONER EN BASE A F QUE INCLUYE A LOS PARAMS DE LA ACCION Y ASI QUEDA TODO MAS COMPACTO]

For a durative action schema ξ, the size of its space of possible action models is then $D \times 2^{5\times|\mathcal{I}_{\xi,\Psi}|}$ where D is the number of different possible durations for any action shaped by the ξ schema. Note that this space is significantly larger than for learning STRIPS actions [21], where this number is $2^{2\times|\mathcal{I}_{\xi,\Psi}|}$ because negative effects must also be preconditions of the same action and cannot be positive effects of that action [PERO LO DE QUE "negative effects must also be preconditions of the same action and cannot be positive effects of that action" NO TIENE NADA QUE VER CON ESE ESPACIO DE POSSIBLE MODELS?? O YO NO LO ENTIENDO??? PORQUE SON $\mathcal{I}_{\xi,\Psi}$ COMO CONDS Y TAMBIEN COMO EFECTOS, BIEN POSITIVOS O NEGATIVOS. OTRA COSA ES QUE REALMENTE ESTO SEA LA COTA SUPERIOR PORQUE SE SABE QUE SI HAY UN EFECTO NEGATIVO TIENE QUE ESTAR COMO PRECONDICION, PERO EL ESPACIO ES ESE QUE SE HA DEFINIDO].

With this vocabulary defined, then the conditions and effects of a durative action schema can be coded by 5 bit-vectors, each of length $|\mathcal{I}_{\xi,\Psi}|$. A 0-bit in the vector represents that the corresponding condition of effect is not part of the schema while a 1-bit represents that is part of the schema. This also means that the *Hamming distance* can be used straightforward as a similarity metric for durative schemes, For instance, to compare a learned action model with respect to a given reference model that serves as baseline. The number of wrong 1-bits in the learned schema provide us a measure of the incorrectness of the learned model and the number of wrong 0-bits in the learned schema provide us a measure of the incompleteness of that model. [ESTE PARRAFO ME GUSTA MUCHO, PERO YO LO MOVERIA A LA SECCION DE EXPERIMENTOS QUE AQUI QUEDA RARO]

Available prior knowledge can be used to bound this vocabulary for certain action schemes. For instance in a given domain we may known in advance several preconditions and/effects of a particular action.

3.2 One-shot learning of concurrent action models

We define 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 where actions in $A?$ are partially specified: the exact conditions/effects, their temporal annotation and/or the duration of actions are unknown. In the worst case, we only know the vocabulary of the symbols that can appear in the conditions/effects of the actions, which means having only the set of candidate conditions/effects.
- O is the set of observations over a plan execution. Al least it contains a full observation of the initial state (time-stamped with $t = 0$) and a final state observation that equals the goals G of the 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] and the times when actions start and/or end their execution. For instance, a partial state can be given by the set of observations $\langle \mathsf{obs}(f_1, t_1), \mathsf{obs}(f_2, t_2) \ldots \mathsf{obs}(f_n, t_n)\rangle$, where each $\mathsf{obs}(f_i, t_i)$ denotes the value observed for fact $f_i \in F$ at time t_i. To simulate the observations of the execution of actions in the plan, we can observe $\langle \mathsf{obs}(is\_start(a_i), t_1), \mathsf{obs}(is\_start(a_j), t_2) \ldots \mathsf{obs}(is\_end(a_k), t_n)\rangle$, representing the fact that action a_i starts at t_1, a_j starts at t_2, and a_k ends at t_n. Figure 2 shows an example of the observation of a plan execution that is taken from the *driverlog* domain.

??

Figure 2. Example of the observation of a plan execution.

- C is a set of *mutex-constraints* that reflects domain-specific expert knowledge. These constraints allow us to deduce new observations to prune inconsistent action models because of mutex (mutual exclusion) information. Figure 3 show an example of a set of state-constraints for the *driverlog* domain.

CREO QUE HABRIA QUE PONER UN EJEMPLO DE LO QUE SON LAS CONDICIONES/EFECTOS CANDIDATOS, POR EJEMPLO A PARTIR DE LAS ACCIONES DE LA FIGURA 1!!! Y LUEGO A CONTINUACION PONER EL EJEMPLO DE LAS STATE-CONSTRAINTS PARA DICHO EJEMPLO QUE INCLUYE AT, DRIVING Y EMPTY

Habria que poner algo en plan $\forall d_1 - driver, y_1, y_2 - location : \ldots$

$\forall x_1, y_1, y_2 : \neg at(x_1, y_1) \vee \neg at(x_1, y_2), \neq (y_1, y_2)$

$\forall x_1, y_1, y_2 : \neg in(x_1, y_1) \vee \neg in(x_1, y_2), \neq (y_1, y_2)$ NO EXPLICADA PORQUE in ES PARA PAQUETE!

$\forall x_1, y_1, y_2 : \neg driving(x_1, y_1) \vee \neg driving(x_1, y_2), \neq (y_1, y_2)$ NOMBRAR MEJOR PARA QUE SE VEA QUE FIJAMOS EL DRIVER

$\forall x_1, y_1, y_2 : \neg driving(y_1, x_1) \vee \neg driving(y_2, x_1), \neq (y_1, y_2)$ NOMBRAR MEJOR LOS PARAMS PARA QUE SE VEA QUE AHORA FIJAMOS EL TRUCK??

$\forall x_1, y_1, y_2 : \neg at(x_1, y_1) \vee \neg driving(x_1, y_2)$

$\forall x_1, y_1, y_2 : \neg at(x_1, y_1) \vee \neg in(x_1, y_2)$ QUITAR PORQUE ES PARA PAQUETE!

$\forall x_1, y_1 : \neg empty(x_1) \vee \neg driving(y_1, x_1)$

$\forall x_1 : \neg link(x_1, x_1)$ [REALMENTE ESTO NO SIRVE COMO MUTEX SINO PARA LA INSTANCIACION]

$\forall x_1 : \neg path(x_1, x_1)$

Figure 3. Examples of state-constraints for the *driverlog* domain. HACE FALTA PONERLOS TODOS CUANDO NO SE HAN DEFINIDO TODOS EN EL EJEMPLO??? YO PONDRIA NOMBRES MAS SIGNIFICATIVOS A LOS PARAMS

A *solution* for the learning task $\mathcal{L}$ is a fully specified model of durative actions $\mathcal{A}$ such that the conditions, effects and duration of its actions are: i) completely specified, i.e. there is no uncertainty about them; and ii) consistent with $\mathcal{L} = \langle F, I, G, A?, O, C\rangle$. [NO SE SI LO DE consistent SE ENTIENDE BIEN AQUI??] Intuitively, we can build on top of the actions in $\mathcal{A}$ a valid plan whose execution starts in I, produces the observations in O, satisfies all constraints in C, and reaches a final state that satisfies G.

4 Learning action models as constraint satisfaction

Given the one-shot learning task $\mathcal{L}$, as defined in subsection 3.2, we automatically create a CSP whose solution induces an action model that solves $\mathcal{L}$. Our CP formulation is solver-independent and it is inspired by previous work on temporal planning as CP [9, 20][2] .

4.1 The CSP variables

For each action a in $A?$ and candidate condition/effect f of a, we create the variables described in Table 1. For simplicity, we model time in $\mathbb{Z}^+$ and bound all maximum times to the makespan t_{end} observed in O. If the observation of t_{end} is unavailable, we consider a long enough domain for time.

[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). But the observations are noiseless, which means that if a value is observed, that is the actual value of that variable.

[2] Contrarily to those works, we now address the inverse task: rather than deducing a plan from a fully known temporal action model, we want to learn a temporal action model from a partial action model and a set of observations.

Table 1. The CSP variables, domains and semantics.

Variable	Domain	Description
1. $\mathsf{start}(a)$	$[0..t_{end}]$	Start time of a (observed or derived value)
2. $\mathsf{end}(a)$	$[0..t_{end}]$	End time of a (observed or derived value)
3. $\mathsf{dur}(a)$	$[0..t_{end}]$	Duration of a
4. $\mathsf{is\_cond}(f,a)$	$\{0,1\}$	1 if f is a condition of a; 0 otherwise
5. $\mathsf{is\_eff}(f,a)$	$\{0,1\}$	1 if f is an effect of a; 0 otherwise
6.1. $\mathsf{req\_start}(f,a)$, 6.2. $\mathsf{req\_end}(f,a)$	$[0..t_{end}]$	Interval when action a requires f
7. $\mathsf{sup}(f,a)$	$\{b_i\} \cup \emptyset$	Supporters for causal link $\langle b_i, f, a\rangle$
8. $\mathsf{time}(f,a)$	$[0..t_{end}]$	Time when the effect f of a happens

The three first variables are self-explanatory and their value is either observed in O or derived by the expression $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. Four and five are decision variables (for readability in comparisons, 1=*true* and 0=*false*): $\mathsf{is\_cond}(f,a)$ models whether f is a condition of a, whereas $\mathsf{is\_eff}(f,a)$ models whether f is an effect of a. The two sixth variables model the $[\mathsf{req\_start}(f,a)..\mathsf{req\_end}(f,a)]$ interval throughout condition f must hold, provided $\mathsf{is\_cond}(f,a)$=*true*. The seventh variable, $\mathsf{sup}(f,a)$, represents the causal link relationship and models the actions b_i that can support f of a. If f is not a condition of a ($\mathsf{is\_cond}(f,a)$=*false*) then $\mathsf{sup}(f,a)=\emptyset$, thus representing an empty supporter. The eighth variable, $\mathsf{time}(f,a)$, models when effect f happens in a, provided $\mathsf{is\_eff}(f,a)$=*true*.

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

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

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

This formulation can also model both TILs and *observations*. On the one hand $\mathsf{til}(f,t)$ are modeled as the dummy action ($\mathsf{start}(\mathsf{til}(f,t)) = t$ and $\mathsf{dur}(\mathsf{til}(f,t)) = 0$) with no conditions and the single effect f that happens at time t ($\mathsf{is\_eff}(f,\mathsf{til}(f,t))$=*true* and $\mathsf{time}(f,\mathsf{til}(f,t)) = t$). On the other hand, $\mathsf{obs}(f,t)$ is modeled as another dummy action ($\mathsf{start}(\mathsf{obs}(f,t)) = t$ and $\mathsf{dur}(\mathsf{obs}(f,t)) = 0$) 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. As can be seen, til is analogous to init, as they both represent information that is given at a particular time, but externally to the execution of the plan. Alternatively, obs is analogous to goal, as they both represent conditions that must be satisfied in the execution of the plan at a particular time.

4.2 The CSP constraints

Table 2 shows the constraints defined among the CSP variables of Table 1. The first two constraints are explicit enough. The third constraint is a double implication that means that when $\mathsf{is\_cond}(f,a)$=*false* it will require no supporter (alternatively, f in a needs a valid supporter only when $\mathsf{is\_cond}(f,a)$=*true*). Constraint four forces to have valid values for the req_start and req_end variables, i.e. the interval condition. The fifth constraint models the causal link $\langle b,f,a\rangle$. Intuitively, the time when b supports f must be before a requires f. Note that in this causal link, $\mathsf{time}(f,b) < \mathsf{req\_start}(f,a)$ and not $\leq$ because, like in PDDL2.1 [7], our temporal planning model assumes an $\epsilon > 0$. The value of ϵ denotes a small tolerance that implies no collision between the time when an effect f is supported and when it is required. When time is modeled in $\mathbb{Z}^+$, $\epsilon = 1$ and $\leq$ becomes $<$. Given a causal link $\langle b,f,a\rangle$, constraint six avoids the threat of action c, which deletes f. It is solved via *promotion* or *demotion* [12], which means bringing $\mathsf{time}(\textit{not-}f,c)$ backward or forward, respectively, in time. The seventh constraint avoids action a from being a supporter of f when $\mathsf{is\_eff}(f,a)$=*false*. Constraint eight models the fact that the same action requires and deletes f; then 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]. The ninth constraint solves the fact that two arbitrary actions have contradictory effects. These nine constraints apply to any type of action, including the dummy actions (init, goal, til and obs). The tenth constraint, however, only applies to non-dummy actions and forces any action to have at least one condition and one effect. We include this constraint to make the learning task more rational, as an action without conditions can be arbitrarily annotated at many times. Alternatively, an action without effects is unnecessary in any plan.

As can be noticed, some conditions of Table 2 are redundant. See for instance constraints five and six: $\mathsf{sup}(f,a) = b$ means obligatorily $\mathsf{is\_eff}(f,b) =$ *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 $\mathsf{is\_cond}(f,a)$=*false*, variables $\mathsf{req\_start}(f,a)$ and $\mathsf{req\_end}(f,a)$ become useless.

4.2.1 *Mutex constraints*

The mutex relationships defined in C for a learning task $\mathcal{L}$ allows us to infer new information in form of dynamic observations that improve the temporal action model.

More specifically, if two variables $\langle f_i, f_j\rangle$ are mutex they cannot hold simultaneously. But it is important to note that, in a rich temporal model, f_i does not necessarily imply *not-*f_j. See action `drive-truck` of Figure 1, where `(at ?t ?l1)` and `(at ?t ?l2)` are mutex as defined in Figure 3. But effects `(not (at ?t ?l1))` and `(at ?t ?l2)` happen *at start* and *at end*, respectively. In other words, clearly the same truck cannot be in two locations simultaneously, but *being* in `?l2` is possible some time after *not being* in `?l1`. Note that this situation does not happen in simple temporal models, or in STRIPS, where all effects happen at the same time and if $\langle f_i, f_j\rangle$ are mutex, f_i implies *not-*f_j and vice versa.

In order to model the mutex constraint between two variables $\langle f_i, f_j\rangle$ we need to create dynamic observations. Roughly speaking, immediately after a asserts f_i, we need to ensure the observation of *not-*f_j. This is done while performing the search, and if $\mathsf{is\_eff}(f_i,a)$ takes the value *true*, then the next observation is added: $\mathsf{obs}(\textit{not-}f_j, \mathsf{time}(f_i,a) + \epsilon)$. Note that the time for 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

Table 2. The CSP constraints and semantics.

Constraint	Description
1. $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$	Relationship among start, end and duration of a
2. $\mathsf{end}(a) \leq \mathsf{start}(\mathsf{goal})$	Always goal is the last action of the plan
3. **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 $\emptyset$
4. **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
5. **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 $b \neq \emptyset$)
6. **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 $b \neq \emptyset$)
7. **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
8. **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
9. **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*)
10. $\sum \mathsf{is\_cond}(f_i, a) \geq 1$ **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

new observation is trivial when using a Dynamic CSP (DCSP), in which the original formulation can be altered. 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 twofold: 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*

Modeling the mutex information increases the CP model size, specially in non-DCSPs, but it can be automated together with the creation of the constraints of Table 2. In practice, the mutex information becomes very useful for learning negative effects. The learning task satisfies the causal links of positive variables, but in the absence of many observations there is no real need to learn negative effects. Mutex information helps to fill this void by inferring the observation of negated variables, which forces to satisfy the causal link of negative variables.

4.2.2 *Constraints for the PDDL2.1 model*

Our temporal planning model is more expressive than PDDL2.1, but we can make it PDDL2.1-compliant by adding the simple constraints of Table 3 for all non-dummy actions. The first constraint limits conditions to be only at *at start*, *over all* or *at end*, whereas the second one limits effects to happen *at start* or *at end*. The third constraint makes the duration of all occurrences of the same action equals. The structure of conditions/effects of all actions $\{a_j\}$ grounded from a particular operator are fixed, so the fourth constraint makes the conditions of all $\{a_j\}$ equal. The fifth constraint is analogous for the effects. Finally, the sixth constraint forces all actions to have at least one of its *n*-effects *at end*. Actions with only *at start* effects would turn the value of the duration irrelevant and they could exceed the plan makespan. Although this constraint is not specific of PDDL2.1, we include it to learn more rational durative actions.

4.3 The CSP cost functions

We want to produce plans that are consist with the input knowledge but that as well provides an *explanation* of the observations that is as tight and lean as possible. To prefer this kind of *tight* and *lean* explanations we define the following two positive functions:

f_1 *Causal-links*. This function counts the number of causal links that are created to support the provided observations.

Table 3. The simple constraints to fulfill a PDDL2.1 model of actions.

Constraint
1.1. ($\mathsf{req\_start}(f, a) = \mathsf{start}(a)$) **OR** ($\mathsf{req\_start}(f, a) = \mathsf{end}(a)$)
1.2. ($\mathsf{req\_end}(f, a) = \mathsf{start}(a)$) **OR** ($\mathsf{req\_end}(f, a) = \mathsf{end}(a)$)
2. ($\mathsf{time}(f, a) = \mathsf{start}(a)$) **OR** ($\mathsf{time}(f, a) = \mathsf{end}(a)$)
3. $\forall a_i, a_j$ occurrences of the same action: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$
4.1. $\forall f_i$: ($\forall a_j$: $\mathsf{req\_start}(f_i, a_j) = \mathsf{start}(a_j)$) **OR** ($\forall a_j$: $\mathsf{req\_start}(f_i, a_j) = \mathsf{end}(a_j)$)
4.2. $\forall f_i$: ($\forall a_j$: $\mathsf{req\_end}(f_i, a_j) = \mathsf{start}(a_j)$) **OR** ($\forall a_j$: $\mathsf{req\_end}(f_i, a_j) = \mathsf{end}(a_j)$)
5. $\forall f_i$: ($\forall a_j$: $\mathsf{time}(f_i, a_j) = \mathsf{start}(a_j)$) **OR** ($\forall a_j$: $\mathsf{time}(f_i, a_j) = \mathsf{end}(a_j)$)
6. $\sum_{i=1}^{n} \mathsf{time}(f_i, a) > n \times \mathsf{start}(a)$

f_2 *Side-effects*. This function counts the number of possitive effects that are added by the actions in a plan but that do not build any causal link.

Our aim is to compute solutions to the CSP that minimize function f_1 while function f_2. is maximized. To achieve this we ask the CSP solve to *pareto optimize* functions f_2 and -f_1(i.e. the negation of function 1).

5 A UNIFIED CP FORMULATION FOR PLANNING, VALIDATION AND LEARNING

Our formulation is connected to the tasks of plan *synthesis* and plan *validation*, and this connection applies not only to temporal planning but also to the classical planning model, the vanilla model of AI planning where actions are instantaneous [10].

The connection between planning, validation and learning tasks lies on the fact that we can restrict the variables of our CP formulation to known values. This feature is useful to leverage a priori knowledge of a given planning domain. For instance, because we have some knowledge about the possible durations of a given action or because we already know that a given action produces for sure certain effects. This approach allows us to synthesize a plan with a given action model. In this case, each variable representing the conditions, effects and duration of the actions are constrained to a single value. Likewise, we can validate a plan by constraining the start times of actions, as we will see in Section 6.

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

- Partial knowledge of the conditions/effects of actions.
- Partial knowledge of actions durations (i.e. a set of possible durations).

- Partial knowledge of the plan to validate or synthesize.

To illustrate this, let us assume that the distribution of all (or just a few) conditions and/or effects is known and, in consequence, represented in the model A? of $\mathcal{L}$. If a solution to the CSP is found, then that structure of conditions/effects is consistent for the learned model. On the contrary, if no solution is found that structure is inconsistent and cannot be explained. We can also represent known values for the durations by bounding the value of $\mathsf{dur}(a)$ variables to a given value. We can also introduce a priori knowledge about plans by bounding the value of the $\mathsf{start}(a)$ variables.

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.

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 4.2.2. We first get the plans for these domains by using five planners (*LPG-Quality* [11], *LPG-Speed* [11], *TP* [15], *TFD* [6] and *TFLAP* [18]), where the planning time is limited to 100s. The actions and observations on each plan are automatically compiled into a CSP learning instance. Then, we run the one-shot learning task to get a temporal action model for each instance, where the learning time is limited to 100s on an Intel i5-6400 @ 2.70GHz with 8GB of RAM. In order to assess the quality of the learned model, we validate each model *vs.* the other models *w.r.t.* the *struct*ure, the *dur*ation and the *struct*ure+*dur*ation, as discussed in section 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

We have presented a purely declarative CP formulation, which is independent of any CSP solver, to address the learning of temporal action models. Learning in planning is specially interesting to recognize past behavior in order to predict and anticipate actions to improve decisions. The main contribution is a simple formulation that is automatically derived from the actions and observations on each

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

plan execution, without the necessity of specific hand-coded domain knowledge. It is also flexible to support a very expressive temporal planning model, though it can be easily modified to be PDDL2.1-compliant. Formal properties are inherited from the formulation itself and the CSP solver. The formulation is correct because the definition of constraints to solve causal links, threats and effect interferences are supported, which avoids contradictions. It is also complete because the solution needs to be consistent with all the imposed constraints, while a complete exploration of the domain of each variable returns all the possible learned models in the form of alternative consistent solutions.

Unlike other approaches that need to learn from datasets with many samples, we perform a one-shot learning. This reduces both the size of the required datasets and the computation time. The one-shot learned models are very good and explain a high number of samples in the datasets used for testing. Moreover, the same CP formulation is valid for learning and for validation, by simply adding constraints to the variables. This is an advantage, as the same formulation allows us to carry out different tasks: from entirely learning, partial learning/validation (structure and/or duration) to entirely plan validation. According to our experiments, learning the structure of the actions in a one-shot way leads to representative enough models, but learning the precise durations is more difficult, and even impossible, when many actions are executed in parallel.

Our CP formulation can be adapted straightforward to address learning, planning or validation tasks within the classical planning model. In this case actions cannot have conditions *overall* or *at end* as well as they cannot have *at start* effects. Therefore the variables representing this kind of information can be removed from the CSP model (or be set to `false`). Further the duration of any action is fixed to one unit [15]. Finally, our CP formulation can be represented and solved by Satisfiability Modulo Theories, which is part of our current work.

REFERENCES

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

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

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

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

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

[14] 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).

[15] 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).

[16] 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).

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

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

[19] 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).

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

[21] 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).

[22] 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).

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

[24] 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).

[25] 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).