One-Shot Learning of Temporal Action Models with Constraint Programming

Antonio Garrido and Sergio Jiménez

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

Abstract. Otros titulos: Learning Temporal Planning Models from Partial Observability by using Constraint Programming; On the Application of Constraint Programming to Learning and Validating; Learning Temporal Planning Models from Partial Observability of Plan Executions
The paper shows that existing CSP compilations for temporal planning can be adapted to build action models for temporal planning.
150–250 words.
3 Mayo abstract 10 mayo full paper 15 pages (references not included)
Paper: 15 pages
Application track - scheduling, rostering and planning; Incluso en Testing and Verification como segunda opcin

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

1 Introduction

Automated planning is the model-based approach for the task of selecting actions that achieve a given set of goals starting from a given initial state. *Classical planning* is the vanilla model for automated planning and it assumes: fully observable states under a deterministic world, instantaneous actions, and goals that are exclusively referred to the last state reached by a plan [6, 7]. Beyond classical planning, there is a bunch of more expressive planning models that relax the previous assumptions to compute more detailed solutions than classical plans [7].

One of these more expressive planning models is *temporal planning* that relaxes the assumption of *instantaneous actions* [4]. Actions in temporal planning are called *durative actions*, meaning that they have duration and that their corresponding conditions/effects that must hold/happen at different times. Actions in a temporal plan can then be executed in parallel and overlapping in several different ways [3]. Further, temporal plans indicate the precise time-stamp when an action starts and ends [10].

Despite the current performance and potential of state-of-the-art planners, its applicability to real-world tasks is still somewhat limited because of the difficulty of specifying correct and complete planning models [13]. The more expressive the

planning model, the more evident becomes this knowledge acquisition bottleneck, which jeopardizes the usability of AI planning technology. This fact is leading to a growing interest in the planning community for the learning of action models [12].

Observations of past behavior provides indirect, but very valuable information to hypothesize the action models, thus helping future planning decisions and recommendations. The learning of action models from observations is also interesting for proactive assistants when recognizing activities of (human or software) agents to assist them in their daily activities and, as the last frontier, to predict and anticipate their needs or actions [16, 1].

The objective of the learning of action models is to compute a set of action's conditions and effects that is *consistent* with some input observations (usually defined as sequences of state changes, world transitions, expert demonstrations or plan traces/logs). Most approaches for learning planning action models from observations are purely inductive and often require large datasets of observations e.g., thousands of plan observations to compute a statistically significant model that minimizes some error metric over the observations [17, 15, 18, 14]. Defining model learning as an optimization task over a set of observations does not guarantee completeness (the model may not explain all the observations) nor correctness (the states induced by the execution of the plan generated with the model may contain contradictory information).

This paper analyses the application of *constraint programming* for the *one-shot learning* of temporal action models that is, the extreme scenario of learning from a single and partial observation of the execution of a temporal plan. While learning an action model for classical planning means computing the action's conditions and effects that are *consistent* with the input observations, learning temporal action models extends this to (1), identify how these conditions and effects are temporally distributed within the action execution and (2), estimate the actions duration.

Most approaches for learning planning action models are purely inductive and often require large datasets of observations e.g., thousands of plan observations to compute a statistically significant model that minimizes some error metric over the observations [17, 15, 18, 14]. Defining model learning as an optimization task over a set of observations does not guarantee completeness (the model may not explain all the observations) nor correctness (the states induced by the execution of the plan generated with the model may contain contradictory information). This paper analyzes the application of *Constraint Programming* for the *one-shot learning* of temporal action models that is, the extreme case of learning action models from a single and partial observation of the execution of a temporal plan.

While learning an action model for classical planning means computing the action's conditions and effects that are *consistent* with the input observations, learning temporal action models extends this to (1), identify how these conditions and effects are temporally distributed within the action execution and (2), estimate the actions duration. As a motivating example, let us assume a logistics plan trace. Loosely speaking, learning the temporal planning model will allow us: i) to better understand the insights of the logistics scenario in terms of what

is possible (or not) and why, because the model is consistent with the observed data; ii) to suggest changes that can improve the model originally created by a human, e.g. re-distributing the action's conditions, provided they still explain the observations; and iii) to automatically elaborate similar models for similar scenarios, such as public transit for commuters, tourists or people in general in metropolitan areas (*a.k.a.* smart urban mobility).

The learning of classical action models has been addressed by different approaches [2] but, to our knowledge, none learns the temporal features. This means that observations may refer to the executions of concurrent actions which makes our approach suitable for learning in multi-agent environments. Further, the paper evidences that the learning of action models strongly resembles the task of synthesizing and validating a plan that is consistent with the input observations. In this work we assume that input observations are noiseless, meaning that if a value is observed, then the observation is correct. Despite the aimed temporal action model is unknown, a set of possible action models (defined as a partially specified action model) is already known and given as input.

As a motivating example, let us assume a logistics plan trace. Loosely speaking, learning the temporal planning model will allow us to: i) better understand the insights of the logistics scenario in terms of what and why is possible; ii) suggest changes that can improve/complete a given model, e.g. re-distributing the action's conditions and effects, provided they still explain the observations; and iii) automatically elaborate similar models for similar scenarios, such as public transit for commuters, tourists or people in general in metropolitan areas (*a.k.a.* smart urban mobility).

2 Background

This section formalizes the planning models we use in the paper.

2.1 Classical Planning

States are represented with a set F of propositional variables. A state s is a full assignment of values to fluents, so $|s| = |F|$ and the size of the state space is $2^{|F|}$.

A *classical planning problem* is a tuple $\langle F, I, G, A\rangle$, where I is a total assignment of the state variables representing the initial state, $G \subseteq F$ is a set of goal conditions over the state variables, and A is the set of actions for updating the state variables. We assume that actions are grounded from given action schemas or operators, as in PDDL (Planning Domain Definition Language [4]).

Each *action* $a \in A$ has a set of preconditions $\mathsf{pre}(a)$ and a set of effects; $\mathsf{pre}(a), \mathsf{eff}(a) \subseteq F$ s.t., $\mathsf{pre}(a)$ must hold before a starts (this is why they are named *pre*conditions), whereas $\mathsf{eff}(a)$ happen when a ends. This way, a is applicable in a state s if $\mathsf{pre}(a) \subseteq s$. When a is executed, a new state, the successor of s, is created that results of applying $\mathsf{eff}(a)$ on s. Typically, $\mathsf{eff}(a)$ is formed by *positive* and *negative/delete* effects. Fig. 2.1 shows an example of two classical

actions, for a *logistics* scenario, taken the *driverlog* domain of the International Planning Competition[1]. Action `board-truck` boards a driver on an empty truck at a given location, with a fixed duration. In `drive-truck` a driver drives a truck between two locations, provided there is a link between them (the duration depends on the locations)

We define a *plan* for a classical planning problem P as an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that induces the *state trajectory* $\langle s_0, s_1, \ldots, s_n \rangle$ such that $s_0 = I$ and a_i $(1 \leq i \leq n)$ is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$. We say that a plan π *solves* P iff $G \subseteq s_n$; i.e. if the goal condition is satisfied in the last state resulting from the application of the plan π in the initial state I.

```
(:action board-truck
  :parameters (?d - driver ?t - truck ?l - location)
  :precondition (and (at ?d ?l) (empty ?t) (at ?t ?l) )
  :effect (and (not (at ?d ?l)) (not (empty ?t)) (driving ?d ?t)))

(:action drive-truck
  :parameters (?t - truck ?from - location ?to - location ?d - driver)
  :precondition (and (at ?t ?from) (link ?from ?to) (driving ?d ?t))
  :effect (and (not (at ?t ?from)) (at ?t ?to)))
```

Fig. 1. Schema for two classical actions from the *driverlog* domain.

2.2 Temporal Planning

A *temporal planning problem* is a tuple $\langle F, I, G, A \rangle$ where F, I and G are defined like in a classical planning problem P, but A represents a set of *durative actions*. There are several options that allow for a high expressiveness of durative actions. An action can have a fixed duration, a duration that ranges within an interval or a distribution of durations (in this work we model time in $\mathbb{Z}^+$). Actions may have conditions/effects at different times, such as conditions that must hold some time before the action starts, effects that happen just when the action starts, in the middle of the action or some time after the action finishes [5].

One of the most popular action models for temporal planning is the slightly restrictive temporal definition of durative actions introduced in PDDL2.1 [4], which defines a durative action as follows:

– $\mathsf{dur}(a)$, a positive value for the action duration.

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

- $\mathsf{cond}_s(a), \mathsf{cond}_o(a), \mathsf{cond}_e(a) \subseteq F$. Unlike the *pre*conditions of a classical action, now conditions must hold before a (*at start*), during the entire execution of a (*over all*) or when a finishes (*at end*), respectively.
- $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$. Now effects can happen *at start* or *at end* of a, respectively, and can still be positive or negative.

Despite in PDDL2.1 *durative actions* are no longer instantaneous, *at start* and *at end* conditions are checked instantaneously. In the same way, *at start* and *at end* effects are instantaneously applied —continuous effects are not considered. With this regard, the semantics of a PDDL2.1 durative action can be defined in terms of two discrete events, $\mathsf{start}(a)$ and $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. *Over all* conditions must hold at any state between $[\mathsf{start}(a)..\mathsf{end}(a)]$. Fig. 2.2 shows an example of two PDDL2.1 durative actions taken the *driverlog* domain of the IPC that extend the classical actions of Fig. 2.1. .

A *temporal plan* is a set of pairs $\langle (a_1, t_1), (a_2, t_2) \dots (a_n, t_n) \rangle$. Each (a_i, t_i) pair contains a durative action a_i and induces two discrete events, $t_i = \mathsf{start}(a_i)$ and $t_i + \mathsf{dur}(a_i) = \mathsf{end}(a_i)$. Though a sequential temporal plan is syntactically possible, it is semantically useless. Consequently, temporal plans are always given as parallel plans. The semantics of temporal actions can be defined in terms of two discrete *events* start_a and end_a, each of which is naturally expressed as a classical action [11]:

$$\mathsf{pre}(\mathsf{start}_a) = \mathsf{pre}_s(a), \quad \mathsf{pre}(\mathsf{end}_a) = \mathsf{pre}_e(a),$$
$$\mathsf{add}(\mathsf{start}_a) = \mathsf{add}_s(a), \; \mathsf{add}(\mathsf{end}_a) = \mathsf{add}_e(a),$$
$$\mathsf{del}(\mathsf{start}_a) = \mathsf{del}_s(a), \quad \mathsf{del}(\mathsf{end}_a) = \mathsf{del}_e(a).$$

Starting temporal action a in state s is equivalent to applying the classical action start_a in s, first verifying that $\mathsf{pre}(\mathsf{start}_a)$ holds in s. Ending a in state s' is equivalent to applying end_a in s', first verifying that $\mathsf{pre}(\mathsf{end}_a)$ holds in s'. The duration $d(a)$ and precondition over all $\mathsf{pre}_o(a)$ impose restrictions on this process: the end of action a has to occur exactly $d(a)$ time units after action a starts and $\mathsf{pre}_o(a)$ has to hold in all states between the start and the end of the action.

A temporal plan $\pi = \{(a_1, t_1), \dots, (a_k, t_k)\}$ solves P if and only if the induced concurrent plan $\pi' = \langle \mathcal{A}_1, \dots, \mathcal{A}_m \rangle$ solves the associated classical planning problem P' and, for each $(a, t) \in \pi$ the overall preconditions hold in the states $s_i, \dots, s_{j-1}$ of the state sequence induced by π', i.e. in all states between actions $\mathcal{A}_i$ and $\mathcal{A}_j$.

PDDL2.2 is an extension of PDDL2.1 that includes the notion of *Timed Initial Literal* [8] ($\mathsf{til}(f, t)$), as a way of defining a fact $f \in F$ that becomes true at a certain time t, independently of the actions in the plan. TILs are useful to define exogenous happenings; for instance, a time window when a warehouse is open in a logistics scenario ($\mathsf{til}(\mathit{open}, 8)$ and $\mathsf{til}(\neg \mathit{open}, 20)$).

2.3 The observation model

From the trace of a temporal plan we can optionally extract a set of partial observations $\mathsf{obs}(f, t)$, denoting the value observed for fact $f \in F$ at time t.

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

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

Fig. 2. Schema for two durative actions from the *driverlog* domain.

Note that we work with partial observability of the temporal plan, as we observe only a few facts at certain times. Obviously, the observability of all facts at all actions' starting/ending times would lead to a full observability of the plan state trajectory.

Our observation model considers also *observed actions* as extra fluents that indicate the action applied in a given state. Further, this observation model can also distinguish between *observable state variables*, whose value may be read from sensors, and *hidden* (or *latent*) *state variables*, that cannot be observed. Given a subset of fluents $\Gamma \subseteq F$ we say that the observation of the execution of a plan π on a planning problem P is a *Γ-observation* iff it only contains observations of fluents in Γ.

Given a set of propositional variables F, a set of *durative actions* A and an a sequence of observations $\mathcal{O} = \langle o_0^o, o_1^o \ldots, o_m^o \rangle$ such that each o_i^o is a set of observed fluents f at a given time-stamp i, i.e., $\mathsf{obs}(f, t_i)$. Let us define, $P_{\mathcal{O}}$ as the temporal planning problem that is built as follows $P_{\mathcal{O}} = \langle F, A, o_0^o, o_m^o \rangle$.

Definition 1 (Explanation). *We say that a plan π* explains *$\mathcal{O}$ iff π is a solution for $P_{\mathcal{O}}$ that is* consistent *with the state trajectory constraints imposed by the sequence of partial states $\mathcal{O}$.*

The *observation* $\mathcal{O}$ is then a sequence of ordered *landmarks* for the $P_{\mathcal{O}}$ classical planning problem [9], because all the literals in the observation must be achieved by any plan that solves $P_{\mathcal{O}}$ and in the same order as are defined in the observation $\mathcal{O}$.

3 Learning Temporal Models

We define the *one-shot* learning of temporal action models as a tuple $\Lambda = \langle \mathcal{M}, \mathcal{O} \rangle$, where:

- $\mathcal{M}$ is the *initial empty model* that contains only the header of each action model to be learned.
- $\mathcal{O}$ is a single learning example or plan observation.

A *solution* to a learning task $\Lambda = \langle \mathcal{M}, \mathcal{O} \rangle$ is a model $\mathcal{M}'$ s.t. there exists a plan computable with $\mathcal{M}'$ that is consistent with the headers of $\mathcal{M}$ and the observed states of $\mathcal{O}$.

3.1 Learning from partially specified action models. A Simple Task?

Once we have learned the preconditions+effects of a classical action, learning a temporal action model may seem, a priori, a straightforward task as it *just* implies to distribute the conditions+effects in time and estimate durations. However, this is untrue.

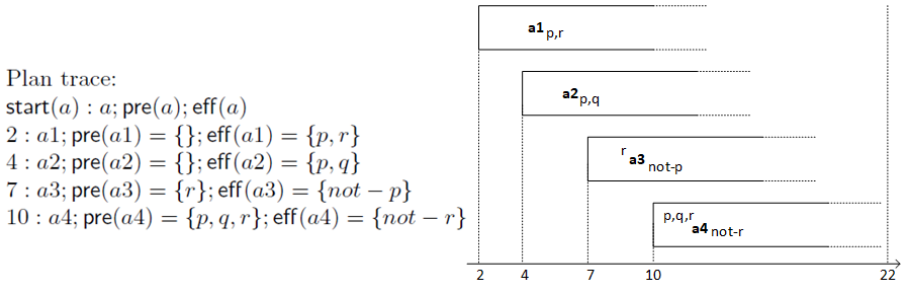

Fig. 3. A simple example of how learning a temporal action model is not straightforward. We know the plan makespan is 22, but depending on the distribution of conditions, effects and durations, many situations are possible, though some of them inconsistent.

Let us suppose the plan trace of Fig. 3, with the preconditions, effects and start times of actions. Clearly, $a3$ needs $a1$ to have r supported, which represents the causal link or dependency $\langle a1, r, a3 \rangle$. Let us imagine that r is in $\mathsf{cond}_s(a3)$. In such a case, if r is in $\mathsf{eff}_s(a1)$, $\mathsf{dur}(a1)$ is irrelevant to $a3$, but if r is in $\mathsf{eff}_e(a1)$, $\mathsf{dur}(a1)$ has to be lower or equal than 5 ($\mathsf{start}(a1) + \mathsf{dur}(a1) \leq \mathsf{start}(a3)$). On the contrary, if r is in $\mathsf{cond}_e(a3)$, $\mathsf{dur}(a1)$ could be much longer. Therefore, the distribution of conditions and effects has a significant impact in the durations, and vice versa.

$a4$ needs p, which means two possible causal links ($\langle a1, p, a4 \rangle$ or $\langle a2, p, a4 \rangle$). The real causal link will be the last to happen, and this depends on the effects+durations of $a1$ and $a2$. Therefore, the causal links are unknown, not

easy to detect and they affect the structure of the temporal plan. But $a4$ really needs both $a1$ and $a2$ to have p, q, r supported. Let us imagine that p, q, r are in $\mathsf{cond}_s(a4)$ and p, q in $\mathsf{eff}_e(a2)$; then $\mathsf{dur}(a2) \leq 6$. Even if we knew for sure that $\mathsf{dur}(a2) = 6$ and r was in $\mathsf{eff}_e(a1)$, we could never estimate the exact value of $\mathsf{dur}(a1)$, as any value in $]0..8]$ would be valid. Intuitively, an action has to wait until the last of its supports, but we cannot grant when the other supports happen; those supporting times and respective durations can never be assured. Therefore, in some situations the precise duration cannot be found and we can only provide values that make the model consistent.

On the other hand, $a3$ deletes p, which means that it might *threat* the causal link $\langle a1, p, a4\rangle$ or $\langle a2, p, a4\rangle$. But again, this threat depends on the distribution of conditions+effects and the durations. For instance, if $not - p$ is in $\mathsf{eff}_s(a3)$, then $a1$ or $a2$ must support p after time 7 and before $a4$ requires it, which entails many consistent alternatives. On the contrary, if p is in both $\mathsf{eff}_s(a1)$ and $\mathsf{eff}_s(a2)$, this plan trace is inconsistent as $a3$ deletes p and no other action in the plan supports p for $a4$. However, if $not - p$ is in $\mathsf{eff}_e(a3)$, $\mathsf{dur}(a3) > 3$ and p is in $\mathsf{cond}_s(a4)$, then no threat will occur in the plan. Therefore, causal links and threats can easily appear or disappear depending on the selected distributions and durations.

Finally, there are some philosophical questions without a clearly motivated answer. First, why some conditions are modeled as *at start* and others as *over all*? In `drive-truck` of Fig. 2.2, why `(driving ?d ?t)` is required throughout the entire action but `(link ?from ?to)` only at its beginning? Apparently, the link between the two locations should remain all over the driving; so is this a wrong decision of the human modeler? Second, why some effects are modeled as *at start* and others as *at end*? In `board-truck`, why is `(not (empty ?t))` happening at start and `(driving ?d ?t)` at end? Could it be in the opposite way? Third, what happens if one action requires/supports what it deletes (see $a4$ in Fig. 3, which might threat itself)? In such a case, the delete effect should happen later than its requirement/supporting. Four, what happens if all effects are *at start*? This makes little sense, as the duration of the actions would be undetermined and could potentially exceed the known plan horizon or makespan no matter the problem goals. In Fig. 3, if the effects of $a1$ and $a2$ are *at start*, is it sensible to allow their durations to pass the limit of 22? In other words, once all plan goals are achieved, can the actions be executed beyond the plan makespan or they need to be cut off to such a value? This decision could potentially lead to an infinite number of models and overlapping situations, so it is not commonly accepted.

As can be noticed from above, learning a temporal action model is not simple. Many possible combinations are feasible provided they fit the constraints the model imposes. Consequently, formulating a CSP from a plan trace and finding a consistent solution that explains that formulation seems an appealing and very promising approach to learn temporal models.

4 One-Shot Learning of Temporal Action Models with Constraint Programming

Our approach is to create a single CSP that includes all the constraints that the observation of the exuction of a temporal plan imposes. This includes: i) actions' conditions, effects, start times and durations; ii) the causal structure of the plan with the supports, to avoid threats and possible contradictory effects; iii) TILs, if exist; and iv) $\mathsf{obs}(f,t)$ observations.

First we define a simple CSP formulation for a temporal model more expressive than PDDL2.1 that allows conditions and effects to be at any time, even outside the execution of the action. For instance, let us imagine a condition p that only needs to be maintained for 5 time units before an action a starts (e.g. warming-up a motor before driving): the expression $\mathsf{req\_end}(p,a) = \mathsf{start}(a); \mathsf{req\_end}(p,a) = \mathsf{req\_start}(p,a) + 5$ is possible in our formulation. We can also represent an effect p that happens in the middle of action a: $\mathsf{time}(p,a) = \mathsf{start}(a) + (\mathsf{dur}(a)/2)$. Then we define the additional constraints that are required for the PDDL2.1 temporal model.

Our CSP formulation is an adaptation of the *planning as CSP* encoding by [5] and is solver-independent. This means that any off-the-shelf CSP solver that supports the expressiveness of our formulation, with binary and non-binary constraints, can be used.

4.1 Variables

For each *action* a in the input observation, we create the seven kinds of variables that are specified in Table 1. Variables define the time-stamps for actions, the causal links, the interval when conditions must hold and the time when the effects happen. To prevent time from exceeding the plan horizon, we bound all times to the makespan of the plan[2].

We create two dummy actions init and goal for each planning problem $\langle F, I, G, A\rangle$:

- init represents the initial state I ($\mathsf{start}(\mathsf{init}) = 0$ and $\mathsf{dur}(\mathsf{init}) = 0$). init has no variables sup, req_start and req_end because it has no conditions. init has as many $\mathsf{time}(p_i, \mathsf{init}) = 0$ as p_i in I.
- goal represents G ($\mathsf{start}(\mathsf{goal}) = makespan$ and $\mathsf{dur}(\mathsf{goal}) = 0$). goal has as many $\mathsf{sup}(p_i, \mathsf{goal})$ and $\mathsf{req\_start}(p_i, \mathsf{goal}) = \mathsf{req\_end}(p_i, \mathsf{goal}) = makespan$ as p_i in G. goal has no variables time as it has no effects.

This formulation allows us to model TILs and observations exactly like other actions. On the one hand, $\mathsf{til}(f,t)$ can be seen as a dummy action ($\mathsf{start}(\mathsf{til}(f,t)) = t$ and $\mathsf{dur}(\mathsf{til}(f,t)) = 0$) with no conditions and only one effect f that happens at time t ($\mathsf{time}(f, \mathsf{til}(f,t)) = t$). A til is analogous to init, as they both represent

[2] We use the makespan, which is known from the input observation, to bound the duration of the actions. This bound is however dispensable considering a long enough domain for durations.

Variable	Domain	Description
$\mathsf{start}(a)$	*known value*	start time of a given in the plan trace
$\mathsf{dur}(a)$	$[1..makespan]$	duration of a. Optionally, it can be bounded by $makespan - \mathsf{start}(a)$
$\mathsf{end}(a)$	*derived value*	end time of a: $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$
$\mathsf{sup}(p, a)$	$\{b_i\}$ that supports p	symbolic variable for the set of potential supporters b_i of condition p of a (causal link $\langle b_i, p, a\rangle$)
$\mathsf{req\_start}(p, a)$, $\mathsf{req\_end}(p, a)$	$[0..makespan]$	interval $[\mathsf{req\_start}(p, a)..\mathsf{req\_end}(p, a)]$ at which action a requires p
$\mathsf{time}(p, a)$	$[0..makespan]$	time when effect p of a happens

Table 1. Formulation of variables and their domains.

information that is given at a particular time, but externally to the execution of the plan.

Last but not least, $\mathsf{obs}(f, t)$ can be seen 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, and no effects at all. An 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 Constraints

Table shows the constraints that we define among the previous variables of Table 1, thus making it possible the learning process from the input observation.

The three first constraints are intuitive enough. The fourth constraint models the causal links. Note that in a causal link $\langle b_i, p, a\rangle$, $\mathsf{time}(p, b_i) < \mathsf{req\_start}(p, a)$ and not $\leq$. This is because temporal planning assumes an $\epsilon > 0$ as a small tolerance between the time when an effect p is supported and when it is required [4]. Since we model time in $\mathbb{Z}^+$, $\epsilon = 1$. This is the reason why $\mathsf{end}(a) < \mathsf{start}(\mathsf{goal})$ and not $\leq$; two consecutive actions need to be separated by an ϵ.

The fifth constraint avoids any threat via promotion or demotion [7]. The sixth constraint models the fact the same action requires and deletes p. Note here the $\geq$ inequality; this is possible because if one condition and one effect of a happen at the same time, the underlying semantics considers the condition is always checked before the effect. The seventh constraint solves the fact that two (possible equal) actions have contradictory effects.

It is important to note that constraints involve any type of actions. Consequently, init, goal, til and obs are subsumed in this formulation.

4.3 Specific Constraints for Durative Actions of PDDL2.1

PDDL2.1 restricts the expressiveness of temporal planning in terms of conditions, effects, durations and structure of the actions. Our temporal formulation

Constraint	**Description**
$\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$	end time of a
$\mathsf{end}(a) < \mathsf{start}(\mathsf{goal})$	any action must end before goal
$\mathsf{req\_start}(p,a) \leq \mathsf{req\_end}(p,a)$	$[\mathsf{req\_start}(p,a)..\mathsf{req\_end}(p,a)]$ must be a valid interval
if $\mathsf{sup}(p,a) = b_i$ then $\mathsf{time}(p,b_i) < \mathsf{req\_start}(p,a)$	modeling causal link $\langle b_i, p, a\rangle$: the time when b_i supports p must be before a requires p
$\forall b_j \neq a$ that deletes p at time τ_j: if $\mathsf{sup}(p,a) = b_i$ then $\tau_j < \mathsf{time}(p,b_i)$ OR $\tau_j > \mathsf{req\_end}(p,a)$	solving threat of b_j to causal link $\langle b_i, p, a\rangle$ being $b_j \neq a$
if a requires and deletes p: $\mathsf{time}(not-p,a) \geq \mathsf{req\_end}(p,a)$	when a requires and deletes p, the effect happens after the condition
$\forall a_i, a_j \mid a_i$ supports p and a_j deletes p: $\mathsf{time}(p,a_i) \neq \mathsf{time}(not-p,a_j)$	solving effect interference (p and $not-p$): they cannot happen at the same time

Table 2. Formulation of the constraints.

is significantly richer than PDDL2.1, but adding constraints to make it fully PDDL2.1-compliant is straightforward.

First, $((\mathsf{req\_start}(p,a) = \mathsf{start}(a))$ OR $(\mathsf{req\_start}(p,a) = \mathsf{end}(a)))$ AND $((\mathsf{req\_end}(p,a) = \mathsf{start}(a))$ OR $(\mathsf{req\_end}(p,a) = \mathsf{end}(a)))$ limits condition p to be *at start*, *over all* or *at end*, i.e. p is in $\mathsf{cond}_s(a)$, $\mathsf{cond}_o(a)$ or $\mathsf{cond}_e(a)$, respectively.

Further, if a condition is never deleted in a plan, it can be considered as an invariant condition for such a plan. In other words, it represents static information. This type of condition is commonly used in planning to ease the grounding process from the operators; e.g. to model that there is a link between two locations and, consequently, a driving is possible, or modeling a petrol station that allows a refuel action in a given location, etc. Therefore, the constraint to be added for an invariant condition p is simply: $((\mathsf{req\_start}(p,a) = \mathsf{start}(a))$ AND $(\mathsf{req\_end}(p,a) = \mathsf{end}(a)))$, i.e. $p \in \mathsf{cond}_o(a)$.

Surprisingly, invariant conditions are modeled differently depending on the human modeler. See, for instance, `(link ?from ?to)` of Fig. 2.2, which is modeled as an *at start* condition despite: i) no action can be planned to delete that link, and ii) the link should be necessary all over the driving. This also happens in the *transport* domain of the IPC, where a refuel action requires to have a petrol station in a location only *at start*, rather than *over all* which makes more sense. This shows that modeling a planning domain is not easy and it highly depends on human's decision. On the contrary, our formulation checks the invariant conditions and deals with them always in the same coherent way.

Second, $((\mathsf{time}(p,a) = \mathsf{start}(a))$ OR $(\mathsf{time}(p,a) = \mathsf{end}(a)))$ makes an effect p happen only *at start* or *at end* of action a, i.e. p is in $\mathsf{eff}_s(a)$ or $\mathsf{eff}_e(a)$. Also, if all effects happen *at start* the duration of the action would be irrelevant and could exceed the plan makespan. To avoid this, for any action a, at least one

of its effects should happen *at end*: $\sum_{i=1}^{n=|\mathsf{eff}(a)|} \mathsf{time}(p_i, a) > n \cdot \mathsf{start}(a)$, which guarantees $\mathsf{eff}_e(a)$ is not empty.

Third, durations in PDDL2.1 can be defined in two different ways. On the one hand, durations can be equal for all grounded actions of the same operator. For instance, any instantiation of `board-truck` of Fig. 2.2 will last 2 time units no matter its parameters. Although this may seem a bit odd, it is not an uncommon practice to simplify the model. The constraint to model this is: $\forall a_i, a_j (a_i \neq a_j)$ instances of the same operator: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$. On the other hand, although different instantiations of `drive-truck` will last different depending on the locations, different occurrences of the same instantiated action will last equal.

In a PDDL2.1 temporal plan, multiple occurrences of `drive-truck(truck1,loc1,loc2,driver1)` will have the same duration no matter when they start. Intuitively, they are different occurrences of the same action, but in the real-world the durations would differ from driving at night or in peak times. Since PDDL2.1 makes no distinction among different occurrences, the constraint to add is: $\forall a_i, a_j (a_i \neq a_j)$ occurrences of the same durative action: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$. Obviously, this second constraint is subsumed by the first one in the general case where all instances of the same operator have the same duration.

Four, the structure of conditions and effects for all grounded actions of the same operator is constant in PDDL2.1. This means that if `(empty ?t)` is an *at start* condition of `board-truck`, it will be *at start* in any of its instantiated actions. Let $\{p_i\}$ be the conditions of an operator and $\{a_j\}$ be the instances of a particular operator. The following constraints are necessary to guarantee equal structure:

$\forall p_i : (\forall a_j : \mathsf{req\_start}(p_i, a_j) = \mathsf{start}(a_j))$ OR $(\forall a_j : \mathsf{req\_start}(p_i, a_j) = \mathsf{end}(a_j))$

$\forall p_i : (\forall a_j : \mathsf{req\_end}(p_i, a_j) = \mathsf{start}(a_j))$ OR $(\forall a_j : \mathsf{req\_end}(p_i, a_j) = \mathsf{end}(a_j))$

And analogously for all effects $\{p_i\}$ and the instances $\{a_j\}$ of an operator:

$\forall p_i : (\forall a_j : \mathsf{time}(p_i, a_j) = \mathsf{start}(a_j))$ OR $(\forall a_j : \mathsf{time}(p_i, a_j) = \mathsf{end}(a_j))$

As a conclusion, in our formulation each action of the plan trace is modeled separately so it does not need to share the same structure or duration of other actions. Moreover, the time-stamps for conditions/effects can be arbitrarily placed inside or outside the execution of the action, which allows for a flexible and expressive temporal model. But, when necessary, we can simply include additional constraints to restrict the expressiveness of the model, such as the ones provided by PDDL2.1.

4.4 Example

As a simple example, we now show a fragment of the formulation for the plan trace depicted in Fig. 3. For simplicity, we only show the variables and constraints for action $a3$, but the formulation is analogous for all other actions.

The variables are: $\mathsf{start}(a3) = 7$; $\mathsf{dur}(a3) \in [1..15]$; $\mathsf{end}(a3) = \mathsf{start}(a3) + \mathsf{dur}(a3)$; $\mathsf{sup}(r, a3) \in \{a1\}$; $\mathsf{req\_start}(r, a3), \mathsf{req\_end}(r, a3) \in [0..22]$; and $\mathsf{time}(not-p, a3) \in [0..22]$. On the other hand, the constraints are: $\mathsf{end}(a3) < \mathsf{start}(\mathsf{goal})$; $\mathsf{req\_start}(r, a3) \leq \mathsf{req\_end}(r, a3)$; if $\mathsf{sup}(r, a3) = a1$ then $\mathsf{time}(r, a1) < \mathsf{req\_start}(r, a3)$;

if $\mathsf{sup}(r, a3) = a1$ then ($\mathsf{time}(not - r, a4) < \mathsf{time}(r, a1)$ OR $\mathsf{time}(not - r, a4) > \mathsf{req\_end}(r, a3)$); and $\mathsf{time}(not - r, a3) \neq \mathsf{time}(r, a1)$.

MOSTRAR RESULTADOS Y VER QUE HAY VARIAS SOLUCIONES CONSISTENTES

4.5 Properties

Soundness of our formulation is guaranteed by the definition of the constraints of Table 2, where all the branching alternatives to solve causal links, threats and effect interferences are supported. Completeness is guaranteed by the complete exploration of the domain of each variable of Table 1 which can return many learned models in the form of consistent alternative solutions.

NO ME TERMINA DE GUSTAR ESTA DEFINICION DE PROPERTIES. IGUAL HABRIA QUE QUITAR ESTA SECCION PORQUE SE QUEDA MUY POBRE

4.6 Implementation. Use of Heuristics for Resolution

Our CSP formulation is automatically compiled from a temporal plan trace, as presented in Fig. 3. The 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.

Our formulation is solver-independent, which means we do not use heuristics that may require changes in the implementation of the CSP engine. Although this can reduce the CSP resolution performance, we are interested in using the solver as a blackbox that can be easily changed with no modification in our formulation. However, we can easily encode standard static heuristics for variable and value selection that help improve efficiency by following the next ordering:

1. Effects (time). For negative effects, first the lower value and for positive effects, first the upper value. This gives priority to delete effects as $\mathsf{eff}_s(a)$ and positive effects as $\mathsf{eff}_e(a)$.
2. Conditions (req_start and req_end). For req_start, first the lower value, whereas for req_end, first the upper value. This gives priority to $\mathsf{cond}_o(a)$, trying to keep the conditions as long as possible.
3. Supporters (sup). First the lower value, thus preferring the supporter that starts earlier in the plan.
4. Duration (dur). First the lower value, thus applying the principle of the shortest actions that make the learned model consistent.

This simple collection of heuristics has been implemented in Choco overriding the default search strategy for the variable and value selectors.

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

5 Evaluation

syntactic evaluation semantic evaluation

Learning can be seen 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 trace to learn the model, i.e. only one sample is used, and no validation step is required.

From a semantic perspective, the success of the learned model can be assessed by analyzing the success ratio of the learned model against all the 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. We 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 Discussion

Indicar que cambios hace falta para que un cp-planner pueda actuar como learner/validator.

7 Conclusions

Comentar que esta formulacin tambien es valida para SMT (SAT MODULO THEORY)

The formulation is a purely declarative representation and is independent of any CSP solver. Consequently, any CSP solver, which uses its own heuristics, can interpret and handle this formulation.

The whole formulation is automatically derived from the planning domain+problem definition, without the necessity of specific hand-coded domain knowledge

On the contrary, our formulation provides a CSP model that is automatically and directly generated from the problem definition

We use the same model of action to represent all elements of temporal planning, including TILs and observations.

The timed initial literals can be interpreted as simple actions that are forced into the respective happenings (rather than selected into them by the planner), whose precondition is true, and whose only effect is the respective literal.

hacer enfasis en que nosotros aprendemos de un unico plan trace (el one-shot) - the learned model must satisfy all the constraints imposed by the observations. Unlike other approaches that need to learn from many plans, we learn the temporal model only from one plan (one-shot learning), which reduces both the size of the required datasets and the computation time to the minimum. In consequence, in this paper we face the learning task by means of constraint programming and propose a simple but effective CSP formulation to learn temporal planning models regarding the partial observations of the execution of a single temporal plan.

References

1. Aineto, D., Jiménez, S., Onaindia, E., Ramírez, M.: Model recognition as planning. In: International Conference on Automated Planning and Scheduling, (ICAPS-19). p. In press (2019)
2. Arora, A., Fiorino, H., Pellier, D., Métivier, M., Pesty, S.: A review of learning planning action models. The Knowledge Engineering Review **33** (2018)
3. Cushing, W., Kambhampati, S., Weld, D.S., et al.: When is temporal planning really temporal? In: Proceedings of the 20th international joint conference on Artifical intelligence. pp. 1852–1859. Morgan Kaufmann Publishers Inc. (2007)
4. Fox, M., Long, D.: Pddl2. 1: An extension to pddl for expressing temporal planning domains. Journal of artificial intelligence research **20**, 61–124 (2003)
5. Garrido, A., Arangu, M., Onaindia, E.: A constraint programming formulation for planning: from plan scheduling to plan generation. Journal of Scheduling **12**(3), 227–256 (2009)
6. Geffner, H., Bonet, B.: A concise introduction to models and methods for automated planning. Synthesis Lectures on Artificial Intelligence and Machine Learning **8**(1), 1–141 (2013)
7. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: theory and practice. Elsevier (2004)
8. Hoffmann, J., Edelkamp, S.: The deterministic part of IPC-4: an overview. Journal of Artificial Intelligence Research **24**, 519–579 (2005)
9. Hoffmann, J., Porteous, J., Sebastia, L.: Ordered landmarks in planning. Journal of Artificial Intelligence Research **22**, 215–278 (2004)
10. Howey, R., Long, D., Fox, M.: Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In: 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004). pp. 294–301 (2004)
11. Jiménez, S., Jonsson, A., Palacios, H.: Temporal planning with required concurrency using classical planning. In: Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS) (2015)
12. Jiménez, S., De la Rosa, T., Fernández, S., Fernández, F., Borrajo, D.: A review of machine learning for automated planning. The Knowledge Engineering Review **27**(4), 433–467 (2012)
13. Kambhampati, S.: Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In: Proceedings of the National Conference on Artificial Intelligence (AAAI-07). vol. 22(2), pp. 1601–1604 (2007)
14. Kucera, J., Barták, R.: LOUGA: learning planning operators using genetic algorithms. In: Pacific Rim Knowledge Acquisition Workshop, PKAW-18. pp. 124–138 (2018)
15. Mourão, K., Zettlemoyer, L.S., Petrick, R.P.A., Steedman, M.: Learning STRIPS operators from noisy and incomplete observations. In: Conference on Uncertainty in Artificial Intelligence, UAI-12. pp. 614–623 (2012)
16. Ramírez, M.: Plan recognition as planning. Ph.D. thesis, Universitat Pompeu Fabra (2012)
17. Yang, Q., Wu, K., Jiang, Y.: Learning action models from plan examples using weighted MAX-SAT. Artificial Intelligence **171**(2-3), 107–143 (2007)
18. Zhuo, H.H., Kambhampati, S.: Action-model acquisition from noisy plan traces. In: International Joint Conference on Artificial Intelligence, IJCAI-13. pp. 2444–2450 (2013)