A Constraint-based Approach to Learn Temporal Features on Action Models from Multiple Plans

Antonio Garrido

VRAIN, Valencian Research Institute for Artificial Intelligence, Universitat Politecnica de Valencia, Camino de Vera s/n, 46022, Valencia, Spain.

Contributing authors: agarridot@dsic.upv.es;

Abstract

Learning in AI planning tries to recognize past conducts to predict features that help improve action models. We propose a constraint programming approach for learning the temporal features, i.e., the distribution of conditions/effects and durations, of actions in an expressive temporal planning model with overlapping actions, which makes it suitable for knowledge-based multi-agent systems. We automatically build a purely declarative formulation that models time-stamps for durative actions, causal link relationships, threats and effect interferences from an arbitrary number of input plans: from just a unique single trace to many. We accommodate different degrees of input knowledge and support a different range of expressiveness, subsuming the PDDL2.1 temporal semantics. The formulation is simple but effective, and is not only valid for learning, but also for plan validation, as shown in its evaluation that returns high precision and accuracy values.

Keywords: Learning durative action models; Temporal planning; Partial observability; Constraint programming

1 Introduction

Automated planning is the model-based approach for the task of selecting the actions that achieve a given set of goals starting from a given initial state. *Classical planning* (*aka* STRIPS planning) assumes fully observable states under a deterministic world, instantaneous actions, and goals that are exclusively referred to the last state reached by a plan [1, 2]. In consequence, a solution plan for classical planning represents a sequence of planning steps where one or more actions are executed in each step. *Temporal planning* extends classical planning with a richer action model that relaxes the strong assumption of instantaneous actions [3], making it more appealing for the real world. Now, actions have temporal features: different durations and conditions/effects that must hold/happen at different times. Consequently, a solution plan for temporal planning problem needs to indicate the precise time-stamp when an action starts and ends [4].

1.1 Motivation through related work

Planning requires the definition of an action model that represents the semantics of actions, in terms of preconditions and effects, to build plans in an efficient way. Action models can be defined by using different languages: STRIPS, Functional Strips, ADL, etc. [2], though the most widely used is PDDL (Planning Domain Definition Language [2, 3]). Despite the potential of state-of-the-art planners, its applicability to the real world is still somewhat limited because of the difficulty and time-consuming of manually specifying sound and complete action models [5, 6], particularly in expressive temporal action languages [7–9]. The more expressive the action model is, the more evident becomes this knowledge-based engineering bottleneck, which jeopardizes the usability of AI planning technology. This has led to a growing interest in the planning community to learn action models, as a knowledge discovery task [10], to improve their quality and reduce the human effort [6, 11]. This paper follows that trend of knowledge engineering, by using constraints, and focuses on model learning, not on plan generation.

The objective of a learning task is to understand and discover action models that are consistent with a set of noiseless observations (defined as some sequence of state changes, input constraints, world transitions, grammars, heuristic rules, expert demonstrations, plan traces/logs, or state-based graphs). Model learning from observation of past behavior provides indirect, but very valuable, information to hypothesize action models.

Learning, as a feature discovery task from raw or empirical data, is specially interesting in general contexts for identifying structures and improving accuracy and diagnostic expertise to remove the need for manual tasks [12–17]. Learning and machine learning have been traditionally seen as classification tasks, including deep learning text classification [18], Natural Language Processing (NLP) [19] and Named Entity Recognition (NER) [17, 20]. These approaches usually work with documents with unformatted text that needs to be processed and tokenized. They use regular expressions, rules and grammars to identify (and label) tokens within the documents. NER has shown successful for information extraction that seeks to locate and classify named entities into pre-defined categories, whereas recent advances in capsule networks are very valuable to establish hierarchies of entities that are semantically equivalent. However, these approaches need datasets with thousands of samples/observations and return poor classifications from small datasets [19].

In the planning context, most approaches for learning action models are purely inductive that also require large datasets, e.g. thousands of plan observations, to compute a statistically significant model that minimizes some error metric over the observations or that uses combinatorial optimization [6, 21–25]. Defining model learning as an optimization task over a large set of observations does not guarantee completeness (the learned model may fail to explain an observation), nor soundness (the states induced by the execution of the plan generated with the model may contain contradictory information). Statistical models might potentially learn a little from each observation, but not a model 100% valid for an individual sample. Further, in many real-world applications it is expensive, or even impossible, to collect large datasets of observations for training [13]. This happens when the number of samples is limited or when a human needs to perform repetitive actions (learning by demonstration). Therefore, reducing the effort to deal with tractable datasets is desirable [13].

Learning an action model for classical planning is a very special type of classification that involves computing the actions (pre)conditions and effects that are consistent with the input observations. Learning classical action models has been addressed by different approaches that use SATisfiabily models, Markov models, Answer Set Programming, genetic algorithms and even planning techniques [26, 27]. Since pioneering learning systems like ARMS [24], we have seen systems able to learn STRIPS action models with quantifiers like in ADL [6, 28], from noisy actions or states [23, 25], from null state information [29], from incomplete domain models [30, 31], from partially observed plan traces [27, 32, 33], or more recently, from graphs that encode the structure of the state space of problem instances [21, 34] (which are later solved by using SAT and Answer Set Programming systems, respectively). Learning the classical action model has proved successful in literature but, to our knowledge, there is a void for temporal models, as none of these systems deals with durative actions to learn their temporal features from multiple observations. In [35], a CP approach for learning from just one observation is used, but it requires additional information on mutual exclusion relationships to help the learning. Consequently, learning the temporal features is the natural evolution of learning classical models, like temporal planning was the evolution of classical planning.

As a motivating example, let us assume a logistics scenario where temporal features are essential. When driving a truck between two locations, the road should be available all over the execution of the action. However, if we use a plane for transportation, we will only need an airport at the beginning+end of the action. Given a full logistics trace, we can deduce the precise duration of an action. If observations are partial, we might only observe when the trip starts, but not the exact time when it ends (traffic congestion may affect). In classical planning none of the previous statements entails difficulty: the distribution of conditions/effects is fixed, as they are always allocated at the start/end of the action, respectively, and all actions have unitary duration, which is not very realistic. In temporal planning, if a human modeler knows the semantics of the logistics actions, learning the temporal features may seem simple. However, without that expert knowledge, if the human has to distribute arbitrarily named conditions $\{p, q, r\}$, effects $\{\neg p, \neg q, s\}$ and estimate valid durations for unknown actions $\{a1, a2, a3\}$ over a set of partial observations, the task becomes much more complex. The learning task induces many assignments that need to fit a complex puzzle, because not all combinations are valid and they depend on the interactions with other actions, observations and constraints.

1.2 Proposed work

Classical model of actions can be retrieved from several ways. Basically, they can be obtained from past experience of domain modelers, from similar planning domains, or automatically learned by using existing approaches, as presented in Section 1.1. Based on such a model, where classical preconditions+effects of actions are known, this paper proposes a novel application of Constraint Programming (CP) for learning the temporal features from observing the execution of multiple temporal plan traces. Learning the temporal features means: i) to identify how action conditions and effects are temporally distributed in the action execution; and ii) to estimate the action duration according to the observed plan traces. In consequence, learning a classical action model decides which are the conditions/effects, whereas learning the temporal features decides when and how they are used. In a schematic way, the outline of our proposal is:

Goal: Transform a classical model of actions into a temporal model of actions by automatically learning the temporal features.

Input: i) a set of classical planning actions with the preconditions+effects, and ii) a non-empty collection of temporal plan traces that come from an unknown fully specified temporal domain. The plan traces only contain the action names and their start time. We assume both the classical actions and the plan traces are noiseless, meaning that what is observed is correct and with no missing actions in the plans. No assumptions on the quality of the plans are required; that is, minimal makespan-plans are not needed. Note that no intermediate state can be recovered: since the duration of the actions and the distribution of preconditions+effects is fully unknown, no intermediate state can be deduced from the traces.

Process (learning task): generation of a CP formulation that transforms the classical model of actions into a temporal model, where actions have duration and conditions+effects are annotated in time. The temporal model of actions must satisfy the observations induced by all (and not only a subset) the temporal plan traces given as input.

Output: a set of fully defined temporal planning actions, where all conditions+effects are annotated in time. Obviously, if an action precondition/effect is missing in the input, it will not be annotated.

Quality indicators: i) similarity measure (precision) between a reference model and the learned one, and ii) validity measure (accuracy) of the learned model to explain a new plan trace.

1.3 Contributions

Our main contribution is a CP formulation for a very expressive temporal planning model, which allows us to represent all Allen's temporal relationships between two temporal actions. Such formulation is inspired by POCL (Partial Order Causal Link [36, 37]) planning, where the causal relationships are explicitly represented via causal links. A causal link consists of two actions and a predicate: the first action achieves the predicate that the second action needs, thus imposing an ordering between the first and the second action.

CP has been previously used to address temporal planning problems. Some approaches create a *"planning problem $\leftrightarrow$ CP formulation"* mapping, under a conservative model of temporal actions [38], or under a very rich model for planning and/or scheduling [39]. Other approaches use CP to map a particular temporal planning problem (e.g. [40] deals with a representation of a scheduling problem, which is mapped as a CP model to be solved by a CP optimizer). Consequently, CP has proved as a good alternative to temporal planners for calculating plans.

Our formulation keeps the philosophy of using CP but, contrarily to [38, 39], we address the inverse task now. Our goal is not to map a temporal planning problem to calculate a plan, but to learn the temporal features of an action model given a plan trace and a partial model of actions. In other words, the input model of actions used in [39] is our output now, as we are interested in learning/playing the designer's role, *w.r.t* the temporal features, rather than in planning.

Another contribution is to learn as much knowledge as possible from the available observations. Rather than using a machine or deep learning classification approach that builds a statistical model that requires huge datasets, we exploit CP inference for reasoning on arbitrary size collections of plan traces. We can learn from just one plan trace or from dozens of traces, and our model satisfies the 100% of the observations given by any trace. The information we require is automatically extracted from the constraints the plan traces impose. Although traces contain actions, which come from action models that are similar to regular expressions, we do not require such expressions. We only need the actions names, which can be considered as simple labels like in NLP or NER approaches.

In our work, we focus on the temporal aspects and rely on the classical preconditions+effects. Although this could seem a limitation of our proposal, there is no need to deliberately learn them again as they can be successfully obtained by existing approaches, previously modeled by a human, extrapolated from similar scenarios, or simply observed. Also, observations now refer to the execution of overlapping durative actions, in opposition to most of the existing approaches that work with sequential non-durative actions. This makes our approach suitable for learning in multi-agent environments, which is also a contribution to deal with realistic scenarios. Finally, the paper evidences that learning temporal features strongly resembles the task of synthesizing and validating a plan that satisfies all the imposed constraints or, in other words, that is consistent with the noiseless input observations. This is our last contribution: a unified CP formulation for both learning and plan validation, which supports temporal models and addresses plan validation of partial action models beyond the functionality of standard validators [4].

The paper is organized as follows. Section 2 introduces the background and terminology used. Section 3 formalizes the learning task of temporal features. In Section 4 we elaborate our CP formulation for learning, presenting the variables, constraints, formal properties and some simple heuristics. It also shows how our formulation is flexible enough to be used for plan validation. The evaluation and experimental results are shown in Section 5. Finally, Section 6 concludes the paper with the lessons learned and proposes some future work.

2 Background and terminology

This section formalizes the *classical* and *temporal* planning models that we follow in this work, as well as the *Constraint Satisfaction Problem.*

2.1 Classical planning

Let F be a set of facts that represent propositional (true/false) variables. Without loss of generality, we assume that F does not contain conflicting variables p and $\neg p$. A state s is a full assignment of boolean values to variables, $\|s\| = \|F\|$, so the size of the state space is $2^{\|F\|}$. Typically, a state is modeled as a set where only the true facts are present, while the false facts are omitted. In our context, a *classical planning problem* is a tuple $\mathcal{CP} = \langle F, I, G, O, A \rangle$, where I is the full initial state, $G \subseteq F$ is a partial set of goal conditions over F[1], O is a set of operators with open parameters, and A is the set of actions grounded from O, as in PDDL.[2]

Each operator $o \in O$ has a set of preconditions and effects to be grounded as facts in actions in A. Consequently, each action $a \in A$ has a set of preconditions $\mathsf{pre}(a)$ and a set of effects $\mathsf{eff}(a) = \{\mathsf{eff}^+(a) \cup \mathsf{eff}^-(a)\}$; $\mathsf{pre}(a), \mathsf{eff}^+(a), \mathsf{eff}^-(a) \subseteq F$. $\mathsf{pre}(a)$ 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

[1] For simplicity, and without lack of completeness, only positive conditions and goals are used, like in the STRIPS definition of PDDL.

[2] Strictly speaking, the actions A could make unnecessary the inclusion of O, but we keep them separately to improve the readability and use of our formalization.

that results of applying $\mathsf{eff}(a)$ on s. Typically, $\mathsf{eff}(a)$ is formed by positive and negative/delete effects ($\mathsf{eff}^+(a)$ and $\mathsf{eff}^-(a)$, which are asserted and retracted, respectively).

Given a classical planning problem $\mathcal{CP}$, we define a *plan trace*, or simply a plan, for $\mathcal{CP}$ as $\Pi_{\mathcal{CP}} = \langle (a_1, t_1), (a_2, t_2) \dots (a_n, t_n) \rangle$. Each (a_i, t_i) pair contains an instantaneous action $a_i \in A$ and the planning step t_i when a_i starts. This action sequence induces a state sequence $\langle s_1, s_2 \dots s_n \rangle$, where each a_i is applicable in s_{i-1}, being $s_0 = I$, and generates state s_i. In every valid plan $G \subseteq s_n$, i.e., G is satisfied in the last state.

2.2 Temporal planning

A *temporal planning problem* is also a tuple $\mathcal{TP} = \langle F, I, G, O, A \rangle$ where F, I and G are defined like in classical planning. O represents the set of *durative operators* that are grounded in the set A of *durative actions*. There are several options that allow for a high expressiveness of durations. First, an operator can have a fixed duration (i.e., all grounded actions from such operator have the same duration) or a duration that depends on the value of the parameters (i.e., different grounded actions from such operator have different duration). Second, conditions/effects are now annotated at different times, such as conditions that must hold some time before the operator starts, effects that happen just when the operator starts, in the middle of the operator or some time after the operator finishes [39].

A popular model for temporal planning is given by PDDL2.1 [3], a language that somewhat restricts temporal expressiveness, which defines a durative action a (strictly speaking, PDDL2.1 defines durative operators) with the following elements:

- $\mathsf{dur}(a)$, a positive value for the action duration.
- $\mathsf{cond}_s(a), \mathsf{cond}_o(a), \mathsf{cond}_e(a)$. 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. In the simplest case, $\mathsf{cond}_s(a) \cup \mathsf{cond}_o(a) \cup \mathsf{cond}_e(a) = \mathsf{pre}(a)$.[3]
- $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$. Now effects can happen *at start* or *at end* of a, respectively, and can still be positive or negative. Again, in the simplest case $\mathsf{eff}_s(a) \cup \mathsf{eff}_e(a) = \mathsf{eff}(a)$.[4]

The semantics of a PDDL2.1 durative action a can be defined in terms of two discrete events, $\mathsf{start}(a)$ and $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. This means that if action a starts on state s, $\mathsf{cond}_s(a)$ must hold in s; and a ending in state s' means $\mathsf{cond}_e(a)$ holds in s'. *Over all* conditions must hold at any state between s and s', i.e., throughout interval $[\mathsf{start}(a)..\mathsf{end}(a)]$. Analogously, *at start* and *at end* effects are instantaneously applied at states s and s', respectively (continuous effects are not considered). Fig. 1 shows durative actions for

[3]In classical planning, $\mathsf{pre}(a) = \{p, \neg p\}$ is contradictory. In temporal planning, $\mathsf{cond}_s(a) = \{p\}$ and $\mathsf{cond}_e(a) = \{\neg p\}$ is a possible situation, though very unusual.

[4]In classical planning, $\mathsf{eff}(a) = \{p, \neg p\}$ is contradictory. In temporal planning, $\mathsf{eff}_s(a) = \{\neg p\}$ and $\mathsf{eff}_e(a) = \{p\}$ is a possible and frequent situation to block/unblock a resource p used within a.

```
(: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. 1 PDDL2.1 schema for two durative actions (operators) from the *driverlog* domain.

`board-truck` and `drive-truck`. `board-truck` has a fixed duration whereas in `drive-truck` the duration depends on the two locations.

PDDL2.2 is an evolution of PDDL2.1 that extends the initial state I with the notion of *Timed Initial Literals*, TILs [41] ($\mathsf{til}(p, t)$ or $\mathsf{til}(\neg p, t)$). A TIL is introduced as a way of asserting or retracting a proposition $p \in F$ at a certain time t, independently of the actions in the plan.[5] TILs are very useful in temporal planning to define exogenous happenings; for instance, a time window when a warehouse is open in a logistics scenario ($\mathsf{til}(open, 8)$ and $\mathsf{til}(\neg open, 20)$).

Given a temporal planning problem $\mathcal{TP}$, we define a *temporal plan trace* for $\mathcal{TP}$ as $\Pi_{\mathcal{TP}} = \langle (a_1, t_1), (a_2, t_2) \ldots (a_n, t_n) \rangle$. Each (a_i, t_i) pair contains a durative action $a_i \in A$ and $t_i = \mathsf{start}(a_i)$. This temporal plan induces a state sequence formed by the union of all states $\{s_{t_i}, s_{t_i+\mathsf{dur}(a_i)}\}$, where there exists a state $s_0 = I$, and $G \subseteq s_{end}$, being s_{end} the last state induced by the plan.

2.3 Constraint Satisfaction Problems

A *Constraint Satisfaction Problem* (CSP) is defined as the tuple $\langle V, D, C \rangle$, where:

- $V = \langle v_1, v_2 \ldots v_n \rangle$ is a set of n finite domain variables.
- $D = \langle D_{v_1}, D_{v_2} \ldots D_{v_n} \rangle$ are the respective domains defining the set of possible values for each variable $v_i \in V$.
- $C = \langle c_1, c_2 \ldots c_m \rangle$ is a set of constraints binding the possible values of the variables in V. Every constraint $c_i \in C$ is defined as a pair $c_i = \langle V_i, R_i \rangle$, where $V_i \subseteq V$ is a subset of $k \leq n$ variables, and R_i is a k-ary relation on the corresponding subset of domains.

An evaluation ev_i satisfies a constraint $c_i = \langle V_i, R_i \rangle$ if the values of ev_i assigned to the variables in V_i satisfy the relation R_i. An evaluation of values to all variables in V is *consistent* if it does not violate any of the constraints in C, i.e., it is a solution for the CSP $\langle V, D, C \rangle$. In absence of a metric over V,

[5] The information in I can be seen as a particular case of TILs with time $t = 0$.

we deal with a pure satisfaction problem. In such a case, many solutions, or different variable assignments that are consistent with the input constraints, are possible and equally valid. This means that no solution is preferred over another.

3 Learning temporal features from multiple plans

3.1 Learning temporal features in a nutshell

Let us assume the two operators `board-truck` and `drive-truck`, given in Fig. 2, from a simple logistics PDDL domain named *driverlog* of the International Planning Competition (IPC).[6] An operator is a schema that comprises the name of the operator, the open typed parameters to be grounded (e.g. `?d`, `?t` and `?l`) and the preconditions+effects in terms of those parameters. `board-truck` boards driver `?d` on an empty truck `?t` at a given location `?l`. In `drive-truck` a truck `?t` is driven between locations `?from` and `?to`, provided there is a link between them. An action is an instantiated operator where all parameters are fully grounded; e.g. `board-truck(driver2 truck1 s2)` is an action grounded from operator `board-truck`.[7] Given the collection of plan traces of Fig. 2, learning the temporal features returns the information given in Fig. 3. For instance, `board-truck(driver2 truck1 s2)` will be executed throughout interval [43..45], as the learned duration is 2. The conditions `(at driver2 s2)` and `(empty truck1)` are learned at interval [43..43], which means they are *start* conditions (cond_s), whereas `(at truck1 s2)` will be at [43..45], which means they are *over all* conditions (cond_o). The effects `(not (at driver2 s2))` and `(not (empty truck1))` are *start* effects (eff_s), as they are learned at interval [43..43], and `(driving driver2 truck1)` is an *end* effect (eff_e), as it is learned at the end of the action, i.e., interval [45..45]. Note that the output is always a fully specified temporal domain, where no precondition or effect remains without its temporal annotation. The temporal features learned from individual actions can be generalized to the subsequent operators, which allows us to define complete temporal action models in PDDL or other variants.

The precision of the learning task depends on the number of the plan traces, but not on their quality. Using just one trace has the drawback of preventing us from having a better global picture of the action model [35]; the more plan traces we use as input the better the learned model will be. Our proposal is flexible and ranges from the most extreme scenario, where just one plan trace is available (*aka* one-shot learning), to the scenario where many traces are available. The learning task does not require minimal plans whatsoever. Our proposal is valid for plans of any makespan.

[6] `www.icaps-conference.org/index.php/Main/Competitions`

[7] PDDL uses the terms *operator* and *action* indistinctly. However, operators are templates with open parameters, whereas actions are instantiated operators where all parameters are grounded.

operator name: **preconditions**: **effects**:	board-truck(?d - driver ?t - truck ?l - location) (at ?d ?l) (empty ?t) (at ?t ?l) (not (at ?d ?l)) (not (empty ?t)) (driving ?d ?t))
operator name: **preconditions**: **effects**:	drive-truck(?t - truck ?from - location ?to - location ?d - driver) (at ?t ?from) (link ?from ?to) (driving ?d ?t) (not (at ?t ?from)) (at ?t ?to)
plan trace 1:	... 43: board-truck(driver2 truck1 s2); board-truck(driver1 truck2 s0) 46: drive-truck(truck1 s2 s1 driver2); drive-truck(truck2 s0 s3 driver1) 60: drive-truck(truck1 s1 s2 driver2); drive-truck(truck2 s3 s0 driver1) 86: drive-truck(truck2 s1 s2 driver1) 88: board-truck(driver2 truck3 s3) ...
plan trace 2:	01: board-truck(driver2 truck1 s0); board-truck(driver1 truck2 s4) 04: drive-truck(truck1 s0 s1 driver2); drive-truck(truck2 s4 s5 driver1) 18: drive-truck(truck1 s1 s5 driver2); drive-truck(truck2 s5 s3 driver1) 32: drive-truck(truck1 s5 s0 driver2); drive-truck(truck2 s3 s1 driver1) ...
... **plan trace *n***:	...

Fig. 2 Input of the learning task: classical planning actions (operators) definition and a collection of n temporal plan traces.

action name:
execution time:
conditions:
 	
effects:
	board-truck(driver2 truck1 s2)
	[43..45] (duration 2)
	(at driver2 s2) [43..43]; (empty truck1) [43..43];
	(at truck1 s2) [43..45]
	(not (at driver2 s2)) [43..43]; (not (empty truck1)) [43..43];
	(driving driver2 truck1)) [45..45]

action name: **execution time**: **conditions**: **effects**:	board-truck(driver1 truck2 s0) [43..45] (duration 2) (at driver1 s0) [43..43]; (empty truck2) [43..43]; (at truck2 s0) [43..45] (not (at driver1 s0)) [43..43]; (not (empty truck2)) [43..43]; (driving driver1 truck2)) [45..45]
action name: **execution time**: **conditions**: **effects**:	drive-truck(truck1 s2 s1 driver2) [46..56] (duration 10) (at truck1 s2) [46..46]; (link s2 s1) [46..56]; (driving driver2 truck1) [46..56] (not (at truck1 s2)) [46..46]; (at truck1 s1) [56..56]
...	...

Fig. 3 Output of the learning task: temporal planning actions definition.

3.2 Learning task. Formalization

Let us assume a set of operators $O?$ that are partially specified because we do not know the exact structure in terms of distribution of conditions/effects nor the duration. Let us also consider a set of facts F that provide a first order logic interpretation over the parameters of operators in $O?$ to instantiate a set of actions $A?$, which are partially specified like in $O?$. In this work, we assume that we know $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$ for each $a \in A?$, but $\mathsf{dur}(a)$ is unknown. Known information can be extracted from the classical version of the planning problem, from prior knowledge we have on the problem, or given by an expert.

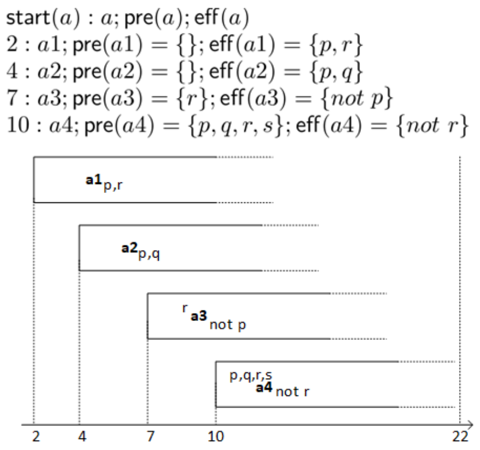
The task of *learning the temporal features on action models* aims at finding a full specified model of $A?$ (and $O?$) from a set of noiseless observed plan traces, which do not need to be of minimal makespan. Consequently, given a collection of n-temporal plan traces $\{\Pi_{\mathcal{TP}_1} \dots \Pi_{\mathcal{TP}_n}\}$, being $n \geq 1$ and where each $\mathcal{TP}_i = \langle F, I_i, G_i, O?, A?\rangle$, we define our learning task of temporal features from n traces as a tuple $\mathcal{L}_n = \langle\{\Pi_{\mathcal{TP}_1} \dots \Pi_{\mathcal{TP}_n}\}, O?, A?\rangle$. The learning task must satisfy the constraints imposed by every $\Pi_{\mathcal{TP}_i}$, so at least one plan trace is needed for reaching some commitments in the learning.

Our definition of learning task requires the minimal amount of observations in each $\Pi_{\mathcal{TP}_i}$, that is, we do not observe any intermediate state in pursuit of minimizing the number of observations. Other learning tasks include intermediate states, require G_i as a full state and input knowledge on static information or mutual exclusion relationships, which make the learning task easier [27, 32, 35]. However, we only require I_i, as a full state, and G_i, as a partial state of goals. It is important to note that, despite knowing the plan traces, we cannot deduce the intermediate states because the structure of the actions is unknown.

$\mathcal{A}$ is a solution to the learning task $\mathcal{L}_n$ if it is consistent with the partial specification given in $A?$, which means having exactly the same conditions and effects, and inducing the temporal plan $\Pi_{\mathcal{TP}_i}$ for all $\mathcal{TP}_i$. $\mathcal{A}$ explains the n plan traces (completeness) and their subjacent temporal model implies no contradictions in the states induced by their executions, where all conditions are satisfied (soundness). Thus, $\mathcal{A}$ is a fully specified model of temporal actions, with all actions of $A?$, where the duration and distribution of conditions+effects is completely specified; i.e., they are all temporally annotated. In other words, for each action $a \in A?$, we have its equivalent version in $\mathcal{A}$ where we have learned $\mathsf{dur}(a)$, $\mathsf{cond}_s(a)$, $\mathsf{cond}_o(a)$, $\mathsf{cond}_e(a)$, $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$. Then, we simply extrapolate the temporal features learned in $\mathcal{A}$ to the operators in $O?$. The distribution of conditions/effects in the operators equals the ones in their grounded actions, but the duration can be fixed per operator or depending on the grounded parameters (see Fig. 1).

3.3 A simple example. Is "learning the temporal features" a simple task?

Learning the temporal features 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 and shows as difficult as solving a non-polynomial search task. Let us consider the simple example of Fig. 4, with the observations on the start times given by just one plan trace $\Pi_{\mathcal{TP}}$, and the model of actions $A?$ for $a1 \dots a4$. Clearly, $a3$ needs $a1$ to have r supported, which represents the causal link or dependency relationship $\langle a1, r, a3\rangle$. 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 ($\mathsf{start}(a1) + \mathsf{dur}(a1) \leq \mathsf{start}(a3)$). From a search perspective, several values for $\mathsf{dur}(a1)$ are possible. On the contrary, if r is in $\mathsf{cond}_e(a3)$, $\mathsf{dur}(a1)$ could be



start(a) : a; pre(a); eff(a)
$2 : a1$; pre$(a1) = \{\}$; eff$(a1) = \{p, r\}$
$4 : a2$; pre$(a2) = \{\}$; eff$(a2) = \{p, q\}$
$7 : a3$; pre$(a3) = \{r\}$; eff$(a3) = \{not\ p\}$
$10 : a4$; pre$(a4) = \{p, q, r, s\}$; eff$(a4) = \{not\ r\}$

Fig. 4 A simple example of how learning the temporal features from a plan trace $\Pi_{\mathcal{TP}}$ and A? is not straightforward. We (optionally) observe the plan makespan is 22 in $\Pi_{\mathcal{TP}}$.

much longer, which means that a higher number of values for dur$(a1)$ are now possible. 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 $\langle a2, p, a4\rangle$), but this depends on the effects+durations of $a1$ and $a2$. This means a new branching point in the search task. Therefore, the causal links are unknown, not easy to detect and they affect the structure of the temporal plan and the complexity of the task. But $a4$ really needs both $a1$ and $a2$ to have p, q, r supported. Let us imagine that p, q, r are in cond$_s(a4)$ and p, q in eff$_e(a2)$; then dur$(a2) \leq 6$. Even if we knew for sure that dur$(a2) = 6$ and r was in eff$_e(a1)$, we could never estimate the exact value of 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 those supports happen. Therefore, in some situations the precise duration cannot be found and we can only provide values that make the model consistent. Clearly, this makes the search task more complex.

$a3$ deletes p, which means that it might *threat* the causal link $\langle a1, p, a4\rangle$ or $\langle a2, p, a4\rangle$. Again, this threat depends on the distribution of conditions+effects and the durations. For instance, if $\neg p$ is in eff$_s(a3)$, then $a1$ or $a2$ must support p after time 7 and before $a4$ requires it, which entails many consistent alternatives. Again, this would mean more branching points. On the contrary, if p is in both eff$_s(a1)$ and eff$_s(a2)$, the observations on this plan trace are inconsistent as $a3$ deletes p and no other action in the plan supports p for $a4$ (no solution exists). However, if $\neg p$ is in eff$_e(a3)$, dur$(a3) > 3$ and p is in 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 rational answer. First, why some conditions are modeled as *at start* and others as *over all*? In `drive-truck` of Fig. 1, why `(driving ?d ?t)` is required throughout the entire action but `(link ?from ?to)` only at its beginning? Formally, the link between the two locations and the fact of being driving must remain all over the action. So is this an irrational or incoherent decision of the human modeler? In Fig. 4, s is a condition that is not asserted nor deleted in the plan so it can be considered as static information, i.e., invariant (*over all*) knowledge that always holds.[8] 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? Although this seems reasonable here, in some domains it is modeled in the other way round. Third, what happens if one action requires/supports what it deletes (see $a4$ in Fig. 4, 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. However, this is possible. In Fig. 4, if the effects of $a1$ and $a2$ are *at start*, is it reasonable to allow their durations to pass a hypothetical limit of 22? In other words, once all plan goals are achieved, can the actions be executed beyond the plan makespan, or do they need to be cut off to such a value? This could potentially lead to an infinite number of models and overlapping situations. Although infrequent, some humans model durative actions only with start effects, which is a bit irrational. Also, temporally expressive examples with required concurrency or single hard envelopes [7, 8], where envelope actions are essential to find a plan, significantly increases the number of overlapping situations, thus increasing the complexity of the search.

As can be noticed, learning the temporal features is not a simple task, and it resembles a non-polynomial task, and many possible combinations are feasible provided they fit the constraints the problem and traces impose. Therefore, a CP formulation seems a promising approach to address this learning task.

4 A CP formulation to learn temporal features on action models

Our approach is to create a CSP that represents the learning task $\mathcal{L}_n = \langle\{\Pi_{\mathcal{TP}_1} \dots \Pi_{\mathcal{TP}_n}\}, O?, A?\rangle$. This comprises two parts. First, to formulate the conditions, effects and durations for actions in $A?$. Second, to formulate: i) the observations of start times for the n plan traces ($n \geq 1$); ii) the causal structure for each trace with all possible supports; and iii) mechanisms to avoid threats and possible contradictory effects in the traces. The advantage here is that we can create a unique formulation by iterating all over the traces, which means that the insights of the formulation are identical no matter the number of input plan traces. Clearly, having more plan traces implies more variables and constraints.

[8]Static information is commonly used in planning for the grounding process, e.g. to model that there is a link between two locations and, consequently, driving between them is possible; to represent that *level1* is before *level2*; to model a petrol station that allows a refuel action in a given location; etc.

Table 1 Formulation of variables and their domains for actions in $A?$. This collection of variables is repeated for each plan $\Pi_{\mathcal{TP}_i}$.

Variable	Domain	Description
$\mathsf{start}(a)$	$[0..\mathit{makespan}]$	start time of a observed in $\Pi_{\mathcal{TP}_i}$
$\mathsf{dur}(a)$	$[0..\mathit{makespan}]$	duration of a
$\mathsf{end}(a)$	$[0..\mathit{makespan}]$	end time of 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 $\langle b_i, p, a\rangle$)
$\mathsf{req\_start}(p, a)$, $\mathsf{req\_end}(p, a)$	$[0..\mathit{makespan}]$	interval $[\mathsf{req\_start}(p, a)..\mathsf{req\_end}(p, a)]$ at which action a requires p
$\mathsf{time}(p, a)$	$[0..\mathit{makespan}]$	time when effect p of a happens

4.1 Variables

For all $\Pi_{\mathcal{TP}_i}$ we first need to create the variables. For each action a in $A?$ that appears in $\Pi_{\mathcal{TP}_i}$, we create the seven kinds of variables specified in Table 1. In our implementation, each variable is annotated with the plan trace it belongs to. For instance, if action a is present in $\Pi_{\mathcal{TP}_i}$ and $\Pi_{\mathcal{TP}_j}$ we create 14 variables (7 per plan). Variables define the time-stamps for actions, the causal links, the interval when conditions must hold and the time when the effects happen. For simplicity, and to deal with integer variables, we model time in $\mathbb{Z}^+$. To prevent time from exceeding the plan horizon, we bound all times to the makespan of each plan $\Pi_{\mathcal{TP}_i}$.[9]

Our temporal model formulation supports all Allen's temporal relationships, so it is more expressive than PDDL2.1 (see more details in Section 4.3), and 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: the expression $\mathsf{time}(p, a) = \mathsf{start}(a) + (\mathsf{dur}(a)/2)$ is also possible.

Additionally, we create two dummy actions init and goal per $\Pi_{\mathcal{TP}_i}$ that correspond with the initial state and goals of the planning problem $\mathcal{TP}_i$. First, init represents the initial state I_i ($\mathsf{start}(\mathsf{init}) = 0$ and $\mathsf{dur}(\mathsf{init}) = 0$). init has no variables $\mathsf{sup}, \mathsf{req\_start}$ and $\mathsf{req\_end}$ because it has no conditions. init has as many $\mathsf{time}(p_j, \mathsf{init}) = 0$ as p_j in I_i. Second, goal represents G_i ($\mathsf{start}(\mathsf{goal}) = \mathit{makespan}$ and $\mathsf{dur}(\mathsf{goal}) = 0$). goal has as many $\mathsf{sup}(p_j, \mathsf{goal})$ and $\mathsf{req\_start}(p_j, \mathsf{goal}) = \mathsf{req\_end}(p_j, \mathsf{goal}) = \mathit{makespan}$ as p_j in G_i. goal has no variables time as it has no effects.

This formulation allows us to model TILs exactly like other actions. $\mathsf{til}(p, t)$ can be seen as a dummy action ($\mathsf{start}(\mathsf{til}(p, t)) = t$ and $\mathsf{dur}(\mathsf{til}(p, t)) = 0$) with no conditions and only one effect p that happens at time t ($\mathsf{time}(p, \mathsf{til}(p, t)) = t$).

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

Table 2 Formulation of constraints. This collection of variables is repeated for each plan trace.

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

As can be noted, a til is similar to init, as they both represent information that is given at a particular time, but externally to the execution of the plan.

4.2 Constraints

Table 2 shows the constraints that we define among the variables of Table 1. In other words, these constraints need to be repeated for the variables created per each plan $\Pi_{\mathcal{TP}_i}$. 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 usually assumes an $\epsilon > 0$, as a small gap to avoid collisions between the time when an effect p is supported and when it is required [3]. When time is modeled in $\mathbb{R}^+$, epsilon is usually 0.001 but when it is modeled in $\mathbb{Z}^+$, $\epsilon = 1$ and $\leq$ becomes $<$. The fifth constraint avoids any threat via promotion or demotion [2]. The sixth constraint models the fact that the same action requires and deletes p. Note the $\geq$ inequality here; this is possible because if one condition and one effect of the same action a happen at the same time, the common underlying semantics in planning considers the condition in a is checked instantly before the effect in a [3]. The seventh constraint solves the fact that two (possibly equal) actions have contradictory effects.

It is important to note that the constraints involve any type of actions. Consequently, init, goal and til 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 and operators. First, we might be interested in conditions/effects that happen at any time of an action a, and not only at $\mathsf{start}(a)$ or $\mathsf{end}(a)$ [9]. Second, PDDL2.1 prevents some action overlapping [7], particularly when scheduling actions with explicit resources [9], because of the gap $\epsilon > 0$. Our temporal formulation subsumes PDDL2.1, so we support conditions/effects at any time thanks to the variables $\mathsf{req\_start}(p, a)$, $\mathsf{req\_end}(p, a)$ and $\mathsf{time}(p, a)$. Also, we can easily modify the granularity of the gap ϵ, which eventually might be zero: $>$ and $<$ inequalities in Table 2 would become $\geq$ and $\leq$, respectively. Consequently, our formulation is very flexible to adopt different temporal models like those presented in [7–9].

Adding constraints to our formulation to make it fully PDDL2.1-compliant is straightforward. First, adding $\mathsf{req\_start}(p, a) = \mathsf{req\_end}(p, a) = \mathsf{start}(a)$ limits condition p to be *at start* (p is in $\mathsf{cond}_s(a)$). Adding $\mathsf{req\_start}(p, a) = \mathsf{start}(a)$ AND $\mathsf{req\_end}(p, a) = \mathsf{end}(a)$ limits condition p to be *over all* (p is in $\mathsf{cond}_o(a)$). Finally, adding $\mathsf{req\_start}(p, a) = \mathsf{req\_end}(p, a) = \mathsf{end}(a)$ limits condition p to be *at end* (p is in $\mathsf{cond}_e(a)$). Furthermore, if a condition p is never deleted in a plan, it can be considered as static or invariant information, and the constraint to be added 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. 1, which is modeled as an *at start* condition despite: i) the link is necessary all over the driving action; and ii) no action in this domain deletes that link. 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 is more rational. This shows that modeling the temporal features is not easy even for an expert and it highly depends on the human's decision. On the contrary, our formulation checks the invariant conditions and deals with them always in a coherent and rational 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*: $\sum_{i=1}^{n=\|\mathsf{eff}(a)\|} \mathsf{time}(p_i, a) > n \times \mathsf{start}(a)$, which guarantees $\mathsf{eff}_e(a)$ is not empty.[10]

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; i.e., fixed duration in the operator. For instance, any grounded action from operator `board-truck` of Fig. 1 will last 2 time units no matter its parameters. Although this may seem a bit odd, it is not an uncommon practice to simplify the model. The constraint to model this is: $\forall a_i, a_j$ being grounded actions of the same operator: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$. On the other hand, although different actions of `drive-truck` could last different depending on the locations, different occurrences of the same 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. They are different occurrences of the same action, but in the real-world the durations would differ from

[10]Certainly, this constraint is not specific of PDDL2.1 but it is required if we want to guarantee *at end* effects like in typical PDDL2.1 domains.

driving at night or in peak times. Since PDDL2.1 makes no distinction among different occurrences, the constraint to add is: $\forall a_i, a_j$ being occurrences of the same 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 actions of the same operator have the same duration.

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

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

As a conclusion, in our formulation each action of $A?$ 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. Also, when necessary, we can simply include additional constraints to restrict the expressiveness of the model, such as the ones provided by PDDL2.1 or other models.

4.4 Result of the formulation

The result of the formulation is a solution to the CSP, as a value for each variable in Table 1 that satisfies all constraints in Table 2. From this solution, learning (and reconstructing) the temporal features for the action schemas, as shown in Fig. 3, is trivial. Let us consider an action a. The temporal features are learned in terms of:

- $\mathsf{start}(a)$, $\mathsf{dur}(a)$ and $\mathsf{end}(a)$, which allow us to learn the execution interval of a in the form $[\mathsf{start}(a)..\mathsf{end}(a)]$.
- Given a condition p, $\mathsf{req\_start}(p, a)$ and $\mathsf{req\_end}(p, a)$ allow us to learn when p is required. For instance, if $\mathsf{req\_start}(p, a) = \mathsf{req\_end}(p, a) = \mathsf{start}(a)$, then p is a *start* condition of a ($\mathsf{cond}_s(a)$). Similarly, we learn *over all* and *end* conditions, i.e., $\mathsf{cond}_o(a)$ and $\mathsf{cond}_e(a)$, respectively.
- Given an effect p, $\mathsf{time}(p, a)$ allows us to learn when p happens. For instance, if $\mathsf{time}(p, a) = \mathsf{end}(a)$, then p is an *end* effect of a ($\mathsf{eff}_e(a)$). Similarly, we learn *start* effects ($\mathsf{eff}_s(a)$).

Extrapolating the temporal features of grounded actions to their corresponding operators, particularly in PDDL2.1, is also trivial as they keep the same structure; e.g., a *start* condition in an action is also a *start* non-grounded condition in its operator.

4.5 Formal properties

Lemma 1 *Soundness. The formulation is sound and a temporal plan defined in terms of durative actions, when all constraints are satisfied, is also sound.*

Proof The lemma relies on the definition of plan soundness, which states that a plan is sound if all conditions of all actions (including the dummy actions) are satisfied at their annotation time, i.e., *at start*, *over all* or *at end*. The formulation provides the same sound scheme defined by POCL planning [36, 37], which is guaranteed by the constraints given in Table 2, including: i) the support of causal links that guarantees that any condition is supported before it is required (fourth constraint); ii) the resolution of threats that ensures that no threatening action can break a causal link (fifth constraint); iii) solving requirements+deletions in the same action (sixth constraint); and iv) avoiding effect interference that ensures that contradictory effects cannot happen simultaneously (seventh constraint). Consequently, all conditions are satisfied, which means the temporal plan is sound and, thereby, the formulation is also sound.

Note that this soundness means that the result of the formulation is a temporal action model that is consistent with a temporal model that is able to generate the input plan traces.

□

Lemma 2 *Completeness. Any solution $\mathcal{A}$ to the formulated learning task $\mathcal{L}_n$ is computable by solving the formulation.*

Proof The formulation creates a CSP that encodes the set of constraints that any solution (i.e., learned model of temporal actions where the duration and distribution of conditions/effects is completely specified) $\mathcal{A}$ must satisfy, and it does not discard any possible model unless it violates any of the constraints of Table 2. By guaranteeing a complete exploration of all variables of the CSP, if a solution exists it can be found, which means that completeness is guaranteed.

Note that completeness means that the result of the formulation is a temporal action model that is consistent not only with a temporal model that is able to generate the input plan traces (soundness), but also with the particular model that generated the input plan traces.

□

Lemma 3 *The number of variables and constraints generated in the formulation is polynomial w.r.t. the number of conditions/effects and actions in the plans.*

Proof Let us consider the formulation for $\mathcal{L}_n = \langle\{\Pi_{\mathcal{TP}_1} \dots \Pi_{\mathcal{TP}_n}\}, O?, A?\rangle$, where there are n plan traces. Let α be the upper bound on the number of actions of each plan $\Pi_{\mathcal{TP}_i}$, and β the upper bound on the number of $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$ for each $a \in A?$. The number of variables is bounded by $\mathrm{O}(n * \alpha * \beta)$. The highest number of constraints depends on the potential number of threats and its solving mechanism (fifth constraint, which involves the preconditions/effects of three actions of the same plan), and it is bounded by O($\alpha * n * \beta^3$). Therefore, the formulation size is polynomial for the variables and constraints. □

It is important to note that these properties are satisfied no matter the specific constraints for PDDL2.1 durative actions are included or not. In other words, PDDL2.1 constraints can limit the types of conditions to three, or the types of effects to two, but they do not jeopardize the soundness, completeness and polynomial size of the formulation.

4.6 Implementation. Use of heuristics for resolution

Constraint satisfaction and propositional SATisfiability are closely related frameworks [42]. Since some of the constraints of our formulation are logical implications that can be turned into conjunctive normal form clauses, using modern SAT solvers seems an interesting option. However, from the point of view of the variables, non-boolean variables make the SAT encoding more tedious (and most of our variables represent time values). For instance, we do not need clauses to ensure that a CSP variable is given a value, nor to ensure that each CSP variable is given only one value [42]. Consequently, we have opted for using a CSP-based implementation rather than a SAT-based one.

We have implemented a compiler in Java that automatically translates the input (operators O?, typically from a planning domain, and the collection of temporal plans) into the CP formulation defined above. If the domain requires PDDL2.1 durative actions, the specific constraints of Section 4.3 are also included in the encoding. As for the solver, we use Choco[11], an open-source Java library for CP that provides an object-oriented API to state the constraints to be satisfied.

In a pure satisfaction problem all possible solutions are equally valid. Although a metric allows the user to specify preferences over the space of solutions, we have not found a metric that guides to the best learning. We have investigated the use of several metrics, e.g. preferring causal links supported by `init` or reducing the number of unused side effects among many others. However, one model is not better than another because it has more or less causal links/effects. In `drive-truck` of Fig. 2, if all locations are connected we could remove the `link` information. Obviously, this new model has fewer causal links and effects, but we cannot ensure that it is a better model, because this depends on the original intention of the human designer. As discussed in Section 3.3, there are many philosophical questions when modeling, and they cannot be represented by simply using a metric. Also, the use of a metric has not a conclusive impact in reducing the variance of the learned model, as this depends on the solver and machine performance. On the contrary, we have opted for defining solver-independent heuristics that are CSP-specific (rather

[11] Choco is available at `www.choco-solver.org`

than SAT-) to guide the search in a univocal way. We use the following standard static heuristics for variable and value selection, which do not require changes in the implementation of the solver engine nor in our formulation:

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 collection of ordering and selection heuristics is very intuitive and has been used in Choco by simply overriding the default search strategy for the variable and value selectors. Although these heuristics have shown very efficient in our experiments, they cannot always guarantee the best performance.

4.7 Using the CP formulation for plan validation

We explained that adding extra constraints allows us to restrict the temporal expressiveness of the learned model. We show here that we can also restrict the learned model by constraining the variables to known values, which is specially interesting when there is additional information on the action model that needs to be represented. For instance, based on past learned models, we may know the precise duration of an action a is 6, or we can figure out that its effect p always happens at end. Our formulation can include this by simply adding $\mathsf{dur}(a) = 6$ and $\mathsf{time}(p, a) = \mathsf{end}(a)$, respectively, which is useful to enrich the partially specified actions in $A?$ of the learning task.

In particular, the possibility of adding those constraints is very appealing when used for validating whether a partial action model allows us to learn a consistent model, as we will see in Section 5. This leads us to a unified formulation for learning and validation. In learning there are many possible values for the formulation variables, whereas in validation the domains are more restricted and some variables are fixed. Let us assume that the distribution of all (or just a few) conditions and/or effects is known and, in consequence, represented in the learning task. If a solution is found, then that structure of conditions/effects is consistent for the learned model. On the contrary, if no solution is found that structure is inconsistent and cannot be explained. Analogously, we can represent known values for the durations. If a solution is found, the durations are consistent, and inconsistent otherwise. Hence, we have three options for validating a partial model *w.r.t.*: i) a known structure with the distribution of conditions/effects; ii) a known set of durations; and iii) a known structure plus a known set of durations (i+ii). The first and second option allows for some flexibility in the learning task because some variables remain open. On the contrary, the third option checks whether a learned model can fit the given constraints, thus reproducing a plan validation task equivalent to [4], but now much more expressive.

5 Evaluation

To the best of our knowledge, there is no approach to learn temporal models (using or not using CP) from an arbitrary collection of plan traces. Therefore, we evaluate our formulation from two independent points of view. First, we evaluate the formulation for the learning task. Second, we assess the learned model by using two quality indicators.

5.1 Setup

We have selected six typical PDDL2.1 temporal planning domains from the IPC, so we have included the constraints of Section 4.3. Although our approach supports very expressive temporal action model domains, we focus on well-known IPC domains. The reason for this is that a great majority of IPC planners are limited in the class of temporal problems they solve [7, 8]; since we use those planners for extensive testing, we are restricted to the domains they support. The number of operators in these domains ranges from 4 (*parking*) to 9 (*rovers*), and the number of propositions per domain ranges from 28 (*zenotravel* and *driverlog*) to 69 (*rovers*). We have used five publicly available planners (*LPG-Quality* [43], *LPG-Speed* [43], *TP* [44], *TFD* [45] and *TFLAP* [46]) to solve different problems per domain. Then we have randomly selected 50 different plans, from the simplest to the most complex ones, which means having 50 plan traces per domain that will be used for learning or testing. It is important to recall that these plans include multiple agents, such as trucks, planes, drivers, rovers, etc., they have overlapping actions, and that we do not need minimal makespan-plans.

We create six learning scenarios $\mathcal{L}_1$, $\mathcal{L}_5$, $\mathcal{L}_{10}$, $\mathcal{L}_{15}$, $\mathcal{L}_{20}$ and $\mathcal{L}_{25}$ with an input collection of 1, 5, 10, 15, 20 and 25 plan traces, respectively. For each scenario we create 25 different problems, which means 25 learning tasks with 1 plan trace ($\mathcal{L}_1$), 25 tasks with 5 plan traces ($\mathcal{L}_5$) and so on. The 25*6=150 learning tasks are solved as satisfaction problems, so we always select the first solution found by using the heuristics of Section 4.6. The runtime was limited to 300s on an Intel i5-6400 @ 2.70GHz with 8GB of RAM.

5.2 Performance evaluation

We measure the 150 learning tasks in terms of their input size and number of variables and constraints of the formulation, as defined in Section 4. Table 3 shows the average values for these terms and includes the learning runtimes per domain in each scenario. For each domain there are three rows. The first row contains the number of actions (A) in the collection of plan traces, the size of the initial state and goals (I and G, respectively) of the planning problem of

Table 3 Size and runtimes of the learning tasks and their formulations for the six scenarios. Symbol '-' means the formulation exceeds the solver capabilities.

	$\mathcal{L}_1$	$\mathcal{L}_5$	$\mathcal{L}_{10}$	$\mathcal{L}_{15}$	$\mathcal{L}_{20}$	$\mathcal{L}_{25}$
	A I G	A I G	A I G	A I G	A I G	A I G
Domain	V C	V C	V C	V C	V C	V C
	time (s)	time (s)	time (s)	time (s)	time (s)	time (s)
	37 220 9	223 1548 54	454 4164 108	659 7528 158	867 11379 210	1103 15477 273
zenotravel	760 6312	4823 47377	10819 111543	17191 173481	24066 236531	31569 311626
	0.03	0.42	1.25	1.68	2.76	3.12
	41 1599 10	287 23598 56	555 81880 114	622 136842 153	-	-
driverlog	2144 12422	27388 158188	88731 441451	145412 668107	-	-
	0.11	2.21	10.23	17.96	-	-
	43 1429 9	226 11384 47	415 32973 89	607 61179 133	800 94992 178	996 130332 226
rovers	2303 12757	15910 80314	41336 195268	73434 336697	111181 500971	150552 671950
	0.25	4.51	15.24	31.15	51.83	80.55
	39 610 15	208 7107 83	383 21035 154	564 39106 226	763 60545 308	965 83424 391
satellite	1160 6041	10047 47143	26469 118587	47089 206982	71347 310736	97094 420706
	0.04	0.46	1.79	3.97	7.05	10.82
	12 255 3	59 1926 15	137 5233 34	197 8382 48	265 12592 65	347 18828 85
storage	501 3842	3133 20719	8042 51428	12432 76626	18038 108599	25996 152127
	0.05	0.15	0.28	0.60	0.87	1.28
	21 457 12	103 3139 62	209 7590 125	311 12345 187	408 17100 249	496 21855 311
parking	845 4432	5106 25231	11573 56012	18294 87590	24902 118095	31350 147371
	0.03	0.14	0.37	0.64	0.96	1.34

the task. The second row contains the number of variables (V) and constraints (C) of the formulation. The third row shows the running time, in seconds, for solving the learning task. For example, in the $\mathcal{L}_1$ scenario of *zenotravel*, the average number of actions is 37, whereas 220 and 9 is the average size of the initial and goal state, respectively. Note that the initial state is always bigger as it represents a full state. The average formulation contains 760 variables and 6312 constraints. The learning runtime is 0.03. Clearly, using a larger input collection of plan traces, e.g. $\mathcal{L}_{25}$ with 1103 actions on average, has a significant impact in both the size of the formulation and the runtime of the learning task. See, for instance, the *driverlog* domain, where the largest learning scenario that can be managed by the solver is $\mathcal{L}_{15}$ with more than 145000 variables and 668000 constraints; dealing with 20 and 25 plan traces in this domain involves an unmanageable CP formulation. The size of the formulation highly depends on the domain complexity, i.e., number of operators, propositions and how they are related. The number of constraints has more impact in the learning runtime than the number of variables (see the *rovers* and *satellite* domains).

5.3 Quality evaluation

The quality evaluation of the learned model 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. The aim is to assess the precision or accuracy of the learned model, a common metric in learning [6, 27, 32, 33]. From a semantic perspective, learning resembles a classification task, where we first learn a model from a training dataset and then validate it on a test dataset. The aim is to assess how much the learned model explains (or transfers) to unseen test samples [13, 47].

5.3.1 Syntactic evaluation. Precision

Precision in learning is a similarity measure between two models, a reference model *vs.* a learned model. In our case, $precision = \frac{p^{=}}{p^{=}+p^{\neq}}$, where $p^{=}$ counts the number of propositions (i.e., conditions+effects) that are temporally distributed equally in both models, and $p^{\neq}$ counts the number of propositions that are distributed in a different way. As an example, given an action or operator, if p is an *at start* condition in the reference model and also in the learned model, it counts as a hit in $p^{=}$, and in $p^{\neq}$ if they differ. This is repeated for all conditions and effects. A precision of 1 means the `:condition` and `:effect` sections in both models are, respectively, syntactically identical. In other words, the learned structure of conditions/effects matches exactly the reference model.

Fig. 5 depicts the precision score for our six learning scenarios. Unfortunately, there is not a unique reference model when learning real-world temporal models; e.g. `full` and ¬`empty` effects can be interchangeable in some domains, both being correct, but they are syntactically different. Also, a pure syntax-based measure may return misleading results, as it may count as incorrect ($p^{\neq}$) a change in the distribution of conditions/effects that represents an equivalent reformulation of the reference model. For instance, given the example of Fig. 1, the condition learned `(over all (link ?from ?to))` would be counted as a difference for action `drive-truck`, as it is *at start* in the reference model. It is specially remarkable that in `drive-truck`, the conditions `(link ?from ?to)` and `(driving ?d ?t)` are *at start* and *over all*, respectively. This is surprising and somewhat irrational, as both of them should be equally annotated as *over all* that leads to a more coherent and sound model. This misleading situation is common in IPC, where the domain definition is not always coherent (even within an operator) due to: i) *at start* conditions that should be *over all* (in *zenotravel*, *driverlog* and *rovers*); and ii) *at start* effects that should be *at end*, and vice versa (in *zenotravel*, *rovers* and *satellite*). In our evaluation, we first use as the reference model the hand-written domain as provided in IPC (REF-model). Then, as a knowledge-based engineering step, we have revised all the domains to fix these irrational situations and recalculate the precision (REV-model) to analyze how the learned models compare to more rational domains.

From Fig. 5 we can conclude that the precision scores are good (above 0.6 except in *rovers* and *parking*), even for the learning scenarios that use a small collection of plan traces as input such as $\mathcal{L}_1$ and $\mathcal{L}_5$. This is an indication that, from our syntax-based experiments, the learned models tend to *converge* with just a few plan traces and dealing with large datasets of traces is not indispensable (note that lines are quite horizontal). Actually, scenarios such as $\mathcal{L}_{20}$ and $\mathcal{L}_{25}$, with big collection of plan traces, show slightly worse syntactic results in the *storage* and *parking* domains because there is more variance. We can also conclude that the learned models are more precise, i.e., they match better, in

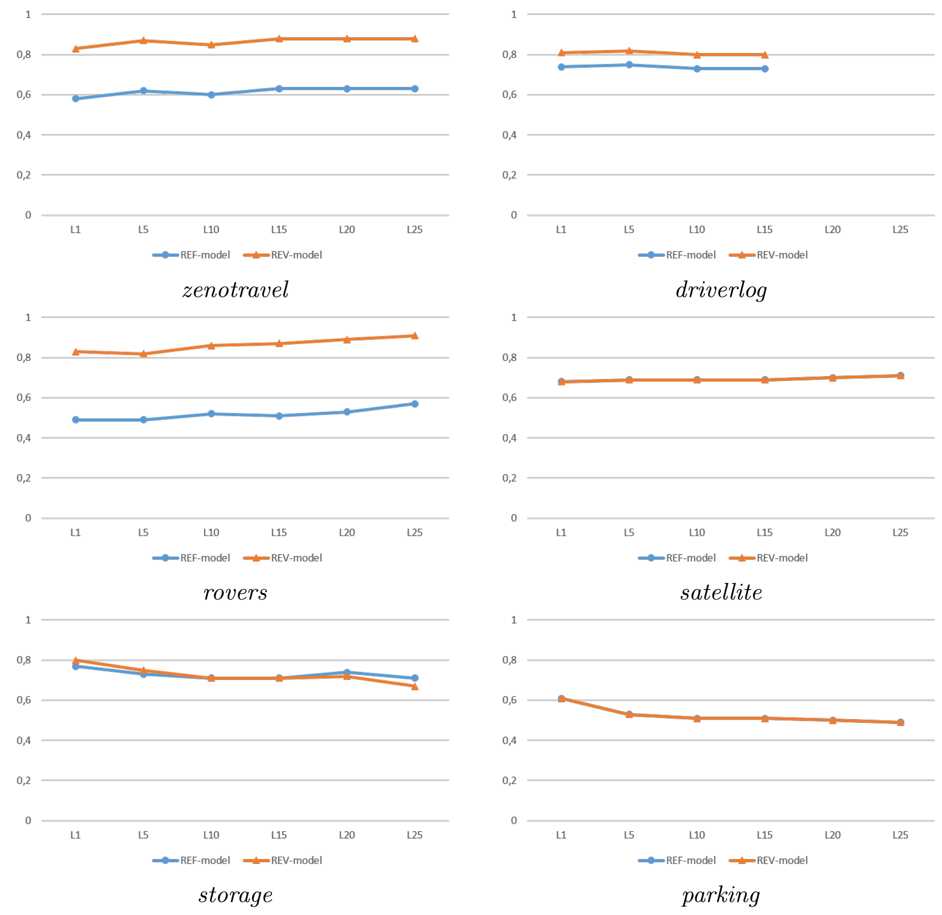


Fig. 5 Precision *w.r.t.* a REFerence and a more coherent and rational REVised model.

the REV-models (specially in *zenotravel* and *rovers*). In *satellite* however, the results remain the same, as the learned models still mislead some effects within the actions. *parking* is a well-defined domain and needs no revision, so the precision is equal in both models. Finally, we have noted that 100% of the *over all* conditions that represent static information (i.e information that always holds) is precisely learned, thus being more coherent than human designers.

5.3.2 Semantic evaluation. Accuracy

As pointed out above, the temporal ground truth about the temporal world is not unique, so similarity measures can be unfair. Also, as seen in Section 3.3, some learned durations cannot be granted and will differ from a reference model, but the underlying model is still consistent. In other words, two syntactically-different models can be semantically equivalent. Therefore, a syntactic evaluation in learning is a bit limited and we should perform a further semantic evaluation. From this standpoint, the quality of the learned model can be assessed by analyzing the accuracy of the learned model *vs.* unseen samples of a test dataset, to check that the learned model is able to explain new plans, analogously to a classification task.

We define the accuracy $Acc = \frac{samples^{\checkmark}}{\|dataset\|}$ as in [31], where $samples^{\checkmark}$ counts the number of samples the learned model explains on a test *dataset*. An accuracy of 1 implies learning a model that explains the full dataset: a feasible solution is found which is consistent with the constraints of the input collection of plan traces together with the test sample trace. But if, for example, one condition is learned as *at start* but it leads to an inconsistency in a test sample (where it must remain *over all*), this does not count in $samples^{\checkmark}$.

In our experiments we have worked with 25 learning tasks per learning scenario, which means learning 25 models. Each learned model is individually validated *vs.* a dataset of 25 new plan traces, resulting in 25*25=625 validation tests per scenario. Each test is repeated three times to validate each model *vs.* the test samples *w.r.t.* the *struct*ure, the *dur*ation and the *struct*ure+*dur*ation, as discussed in Section 4.7. The struct value means that the distribution of conditions/effects learned is consistent with the samples. The dur value means the duration learned is consistent with the samples. The struct+dur value means that the learned model explains entirely the samples. This value is always the lowest because a subtle change in the structure or duration learned that leads to inconsistency counts as a failure. The results depend on the domain complexity, the relationships (causal links, threats and interferences) among the grounded actions, and the size and quality of the plan traces. Fig. 6 shows the average accuracy for our experiments. For instance, in *zenotravel*, the struct values denote the perfect result in the six learning scenarios (accuracy=1). The dur values are also very good starting from the $\mathcal{L}_{10}$ scenario, as also happens in struct+dur (accuracy around 0.9). Learning the duration seems more effective than learning the structure in the *rovers* and *storage* domains, starting from $\mathcal{L}_{10}$ and $\mathcal{L}_{15}$. In *driverlog*, *satellite* and *parking* the results for struct and dur are very similar starting from $\mathcal{L}_5$.

From Fig. 6 we can conclude that there is not a clear answer whether learning the structure/duration in itself is more accurate, as this depends on the domain. We can observe that increasing the size of the input collection of plan traces shows beneficial for the accuracy of the learned model, in particular for the struct+dur values. Starting from the $\mathcal{L}_{15}$ scenario, the values for struct+dur are all over 0.8 except in *storage*. The results are specially remarkable in *driverlog*, *satellite* and *parking* domains. As we might expect, the higher the number of plan traces in the input collection, the more accurate is the learned model to explain unseen samples.

6 Conclusions and lessons learned

The interest in learning is growing up because it allows us to acquire procedural knowledge through demonstration and partial observations. Learning

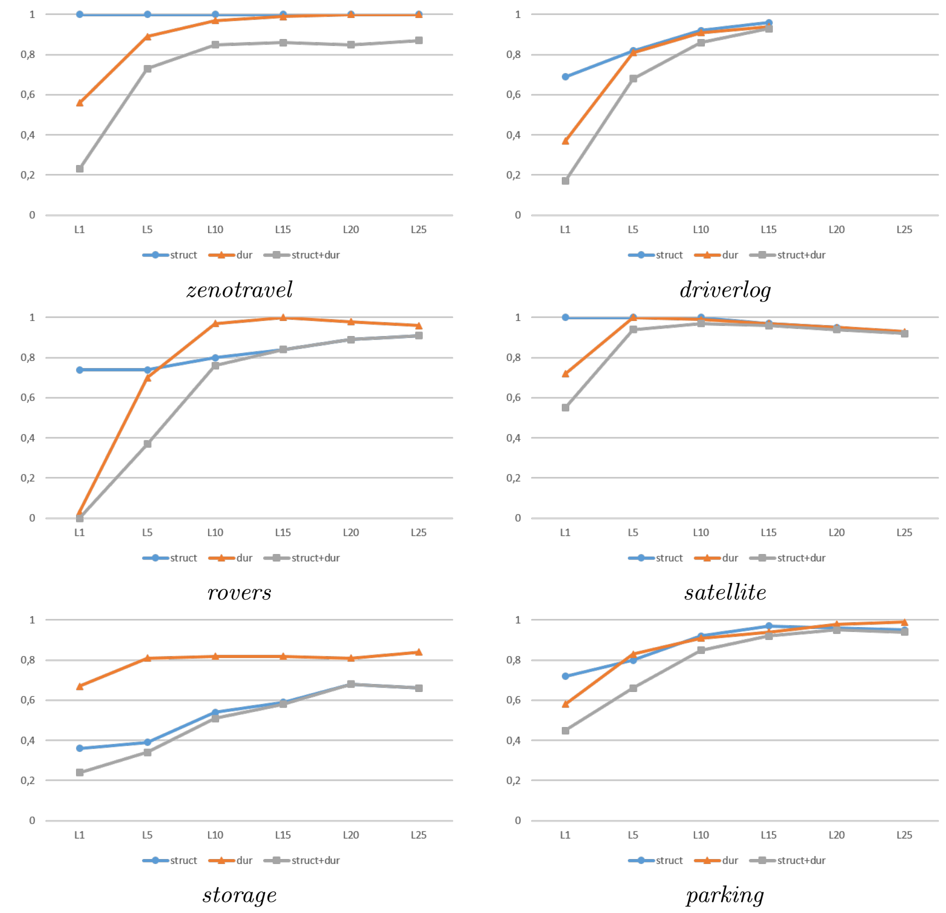


Fig. 6 Average accuracy of the learned model *vs.* the test dataset.

planning action models by observations of plan traces is useful in many scenarios: recognition of past behavior for prediction and anticipation, decision taking and recommendation, programming and modeling, robotics motion capturing and planning, etc. Learning is appealing because these scenarios include a huge number of tasks, sometimes difficult to be described formally (and more difficult to be annotated temporally), which require expert knowledge and engineering that becomes impractical in complex domains.

In this paper we have presented a (solver-independent) purely declarative constraint-based formulation to address the automated learning of temporal features on action models which, to our knowledge, initiates a novel approach for the intersection of knowledge-based systems, learning from multiple plans, CP and planning. We hope the paper will also initiate new lines of research to address the task of learning in temporal planning settings and make further comparisons. Learning the classical model is to planning what learning the temporal features is to temporal planning. Knowing these features is useful in practice, where learning a classical planning model is not enough and needs to be extended with temporal features to increase its applicability. In consequence, learning the temporal features bridges the gap between the (planning) action model and the temporal world. The ultimate goal of learning is to reduce the laborious modeling stage, to minimize the effort human experts need to design temporal planning models before launching the planners.

Our main result is an effective formulation that is automatically derived, without the necessity of specific hand-coded domain knowledge. This formulation has a series of advantages. First, it is flexible enough to learn from multiple plans: from a single plan trace to a large collection of traces. This is useful to manage domains with different levels of input knowledge and complexity; for instance, we have been able to solve learning tasks with more than 50 input plans in the *zenotravel* domain (although these experiments are not presented in the paper), but in more complex domains such as *driverlog* we can still effectively learn from fewer input plans. Second, the quality of the input plans is neither relevant for the learning task nor the theoretical results. The approach learns in terms of the causal links and the constraints given by the plan traces, so optimal plans have no impact in the learning. Third, it does not need intermediate states, but only the initial full state and a partial goal state. Fourth, it supports a rich temporal planning model with high levels of concurrency; although its expressiveness is beyond PDDL2.1, it can be easily modified to be PDDL2.1-compliant. Fifth, formal properties are inherited from the (POCL) formulation itself and the solver. The formulation is sound 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.

Several lessons can be learned from our work. In our context, learning is a task to understand and acquire knowledge that fits many constraints from the input plans, no matter their quality. Therefore, constraint technology is very adequate for this. Unlike other approaches that obligatorily need to learn from datasets with hundreds of traces, we can easily adapt to the number of available traces: from only one to many. Moreover, we can easily deal with more input information by simply including more input observations in our learning task, i.e., by using very long plan traces. Learning from collections of many traces can lead to better models, but retrieving many samples is not always easy, specially in human interactive environments that require learning by demonstration. Using fewer traces reduces the size of the required datasets and formulation, and the computation time. From our experiments, we have learned that a high number of input traces has no a special impact in the syntactic evaluation, but it has a good impact in the semantic evaluation. Using a pure satisfaction problem can lead to many solutions. Ordering heuristics, which are easily reproducible, provide a simple way for breaking ties and guide the search towards the same solution, no matter the solver. Obviously, if many solutions are returned we can always select the most learned model, that is, the most repeated one. We have investigated this, but the most repeated model is not always the best model and this depends on the domain used. An interesting lesson learned is that the same CP formulation is valid for learning and for validation, by simply adding constraints to the variables. This is an innovative 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.

Finally, it is important to note that our formulation can be represented and solved by Satisfiability Modulo Theories, working with the theory of integers, thus overcoming the limitations of the SAT encodings for non-boolean variables. This is part of our current work. As for future work, we want to extend our formulation to learn from intermediate observations (we need to investigate how many and how frequent they must be), to learn meta-models (as combinations of several learned models), and to learn more complete action models trying to find further similarities with NLP and NER approaches. In the latter, we will relax the input action model to find out the conditions/-effects together with their temporal distribution when learning from multiple plans.

Acknowledgments. This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R.

Conflict of Interest - None.

References

[1] 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)

[2] Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Elsevier (2004)

[3] Fox, M., Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research **20**, 61–124 (2003)

[4] Howey, R., Long, D., Fox, M.: VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In: Proc. of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-04), pp. 294–301 (2004)

[5] Kambhampati, S.: Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In: Proc.

of the National Conference on Artificial Intelligence (AAAI-07), vol. 22(2), pp. 1601–1604 (2007)

[6] Zhuo, H.H., Yang, Q., Hu, D.H., Li, L.: Learning complex action models with quantifiers and logical implications. Artificial Intelligence **174**(18), 1540–1569 (2010)

[7] Cushing, W., Kambhampati, S., Weld, D.S., *et al.*: When is temporal planning really temporal? In: Proc. of the International Joint Conference on Artificial Intelligence, pp. 1852–1859 (2007). Morgan Kaufmann Publishers Inc.

[8] Coles, A., Fox, M., Halsey, K., Long, D., Smith, A.: Managing concurrency in temporal planning using planner-scheduler interaction. Artificial Intelligence **173**(1), 1–44 (2009)

[9] Rintanen, J.: Models of action concurrency in temporal planning. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-2015), vol. 99, pp. 1659–1665 (2015)

[10] Baker, C.L., Saxe, R., Tenenbaum, J.B.: Action understanding as inverse planning. Cognition **113**, 329–349 (2009)

[11] 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)

[12] Appiah, A.Y., Zhang, X., Ayawli, B.B.K., Kyeremeh, F.: Long short-term memory networks based automatic feature extraction for photovoltaic array fault diagnosis. IEEE Access **7**, 30089–30101 (2019)

[13] Jialin Pan, S., Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering **22(10)**, 1345–1359 (2010)

[14] Kou, G., Yue, J., Ma, Y., Wang, Q., Zhang, Z.: SAR image invariant feature extraction by anisotropic diffusion and multi-gray level simplified PCNN. IEEE Access **7**, 47135–47142 (2019)

[15] Lauretti, C., Cordella, F., Ciancio, A.L., Trigili, E., Catalan, J.M., Badesa, F.J., Crea, S., Pagliara, S.M., Sterzi, S., Vitiello, N., Garcia Aracil, N., Zollo, L.: Learning by demonstration for motion planning of upper-limb exoskeletons. Frontiers in Neurorobotics **12**, 1–5 (2018)

[16] Liu, L., Wang, S., Hu, B., Qiong, Q., Rosenblum, D.S.: Learning structures of interval-based bayesian networks in probabilistic generative model for human complex activity recognition. Pattern Recognition **81**, 545–561 (2018)

[17] Xu, Y., Tsujii, J., Chang, E.I.-C.: Named entity recognition of follow-up and time information in 20000 radiology reports. Journal of the American Medical Informatics Association **19**(5), 792–799 (2012)

[18] Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J.: Deep learning based text classification: a comprehensive review. https://arxiv.org/pdf/2004.03705.pdf

[19] Zhao, W., Peng, H., Eger, S., Cambria, E., Yang, M.: Towards scalable and reliable capsule networks for challenging NLP applications. In: Proc. of the Annual Meeting of the Association for Computational Linguistics, pp. 1549–1559 (2019)

[20] Zhong, X., Cambria, E., Hussain, A.: Extracting time expressions and named entities with constituent-based tagging schemes. Cognitive Computation **12**, 844–862 (2020)

[21] Bonet, B., Geffner, H.: Learning first-order symbolic representations for planning from the structure of the state space. In: Proc. of the European Conference on Artificial Intelligence (ECAI-2020), pp. 2322–2329. IOS Press (2020)

[22] Kucera, J., Barták, R.: LOUGA: learning planning operators using genetic algorithms. In: Proc. of the Pacific Rim Knowledge Acquisition Workshop (PKAW-18), pp. 124–138 (2018)

[23] Mourão, K., Zettlemoyer, L.S., Petrick, R.P.A., Steedman, M.: Learning STRIPS operators from noisy and incomplete observations. In: Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI-12), pp. 614–623 (2012)

[24] Yang, Q., Wu, K., Jiang, Y.: Learning action models from plan examples using weighted MAX-SAT. Artificial Intelligence **171**(2-3), 107–143 (2007)

[25] Zhuo, H.H., Kambhampati, S.: Action-model acquisition from noisy plan traces. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-13), pp. 2444–2450 (2013)

[26] Arora, A., Fiorino, H., Pellier, D., Métivier, M., Pesty, S.: A review of learning planning action models. The Knowledge Engineering Review **33**, (2018)

[27] Aineto, D., Jiménez, S., Onaindia, E.: Learning action models with minimal observability. Artificial Intelligence **275**, 104–137 (2019)

[28] Amir, E., Chang, A.: Learning partially observable deterministic action

models. Journal of Artificial Intelligence Research **33**, 349–402 (2008)

[29] Cresswell, S.N., McCluskey, T.L., West, M.M.: Acquiring planning domain models using LOCM. The Knowledge Engineering Review **28**(2), 195–213 (2013)

[30] Zhuo, H.H., Kambhampati, S.: Model-lite planning: case-based vs. model-based approaches. Artificial Intelligence **246**, 1–21 (2017)

[31] Zhuo, H.H., Nguyen, T.A., Kambhampati, S.: Refining incomplete planning domain models through plan traces. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-13), pp. 2451–2458 (2013)

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

[33] Zhuo, H.H., Muñoz-Avila, H., Yang, Q.: Learning hierarchical task network domains from partially observed plan traces. Artificial Intelligence **212**, 134–157 (2014)

[34] Rodriguez, I., Bonet, B., Romero, J., Geffner, H.: Learning first-order representations for planning from black box states: New results. In: Proc. of the 18th Int. Conference on Principles of Knowledge Representation and Reasoning, pp. 539–548 (2021)

[35] Garrido, A., Jimenez, S.: Learning temporal action models via constraint programming. In: Proc. of the European Conference on Artificial Intelligence (ECAI-2020), pp. 2362–2369. IOS Press (2020)

[36] McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proc. 9th Nat. Conference on AI, Anaheim, CA, pp. 634–639 (1991)

[37] Penberthy, J., Weld, D.S.: UCPOP: a sound, complete, partial-order planner for ADL. In: Proc. Int. Conference on Principles of Knowledge Representation and Reasoning, pp. 103–114. Kaufmann, Los Altos, CA (1992)

[38] Vidal, V., Geffner, H.: Branching and pruning: an optimal temporal POCL planner based on constraint programming. Artificial Intelligence **170**, 298–335 (2006)

[39] 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)

[40] Booth, K.E.C., Minh Do, J., Beck, C., Rieffel, E.G., Venturelli, D., Frank, J.: Comparing and integrating constraint programming and temporal planning for quantum circuit compilation. In: Proc. of the International Conference on Automated Planning and Scheduling (ICAPS-18), pp. 366–374 (2018)

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

[42] Walsh, T.: SAT v CSP. In: Proc. of the Int. Conference on Principles and Practice of Constraint Programming (CP-2000), pp. 441–456 (2000)

[43] Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs in LPG. Journal of Artificial Intelligence Research **20**, 239–290 (2003)

[44] Jiménez, S., Jonsson, A., Palacios, H.: Temporal planning with required concurrency using classical planning. In: Proc. of the International Conference on Automated Planning and Scheduling (ICAPS-15) (2015)

[45] Eyerich, P., Mattmüller, R., Röger, G.: Using the context-enhanced additive heuristic for temporal and numeric planning. In: Proc. of the International Conference on Automated Planning and Scheduling (ICAPS-09) (2009)

[46] Marzal, E., Sebastia, L., Onaindia, E.: Temporal landmark graphs for solving overconstrained planning problems. Knowledge-Based Systems **106**, 14–25 (2016)

[47] Zhuo, H.H., Yang, Q.: Action-model acquisition for planning via transfer learning. Artificial Intelligence **212**, 80–103 (2014)