Garrido, A. (2022). Learning temporal action models from multiple plans: A constraint satisfaction approach. Engineering Applications of Artificial Intelligence. 108:1-14. https://doi.org/10.1016/j.engappai.2021.104590

# Learning Temporal Action Models From Multiple Plans: a Constraint Satisfaction Approach

Antonio Garrido

*VRAIN, Valencian Research Institute for Artificial Intelligence. Universitat Politecnica de Valencia, Spain. e-mail: agarridot@dsic.upv.es*

**Abstract**

Learning, as a discovery task from past observations, is interesting in engineering contexts for identifying structures and improving accuracy. Learning in planning scenarios aims at recognizing past behavior to predict action models to improve decisions. This is appealing because practical scenarios are usually complex, sometimes difficult to be described formally, which require expert knowledge and engineering that becomes impractical in real-world applications. We introduce a Constraint Satisfaction formulation for learning PDDL2.1 temporal action models in planning. Given a collection of observations on multiple plans and a set of empty operators, we automatically create a learning task that identifies which conditions+effects are necessary, together with their temporal annotation, and induces durations and costs. Our formulation encapsulates planning (causal links, threats and effect interferences) and mutex (to avoid contradictions) constraints to be fully satisfied from the observed plans. The formulation is simple, but it proves very effective and easily adaptable to different levels of expressiveness. We evaluate such effectiveness in different IPC domains and compare the quality of the learning *vs.* other state-of-the-art learning approaches.

*Keywords:* learning action models, planning, temporal planning, constraint satisfaction

## 1. Introduction

Learning, as a feature discovery task from raw or past empirical data, is appealing in academic and industrial environments to identify common structures/behaviors that lead to improvements in diagnostic expertise and reduce the need for manual operations [1, 2, 3, 4, 5, 6].

The use of AI planning, defined as the task that elaborates a plan of actions to reach certain goals, is crucial in industrial applications (e.g. public transportation, logistics, motion planning, robotics and autonomous systems, cognitive assistants, data analysis, plant control, cybersecurity, service composition, etc. [7, 8]). Focusing on AI planning, learning is specially interesting to recognize past behavior in order to anticipate action models that improve knowledge and help further decisions.

In order to build plans, automated planning relies on an action model that captures the physics of the problem, in terms of which conditions are needed for an action to be executed and which effects are achieved once the action is executed. Metaphorically speaking, the conditions and effects would resemble the left- and right- hand sides of a rule, respectively, in a rule-based system. Action models are defined by using different languages, such as STRIPS (Stanford Research Institute Problem Solver [9]), Functional Strips [10] that introduces function symbols that can be nested and provide more efficient encodings, ADL (Action Description Language [11]) with conditional effects and quantification, or PDDL (Planning Domain Definition Language [12]). PDDL is the *de facto* standard language for classical planning, where all actions are instantaneous and only have one type of conditions and effects. Although PDDL models are very useful, there are difficulties and limitations to model real-world problems because they cannot reason on temporal and resource-intensive features. For instance, if we are modeling a logistics scenario, a boarding action will have a different duration than flying between two remote cities. Also, locking and unlocking a shared resource within the same action is impossible, as they represent two contradictory effects.

PDDL2.1 [13] extends PDDL to deal with a more expressive version for temporal planning that meets the challenges of real applications and overcomes the above PDDL limitations. In PDDL2.1, actions are durative, overlap in different ways, and conditions+effects are temporally annotated (i.e., there are three types of conditions and two types of effects, which show much more flexible than in PDDL).

Defining action models (to represent planning domains) from scratch is often a challenge; it is a difficult, error-prone and time-consuming engineering task [14, 15], even for domain experts. There are many reasons for this challenge [16]. Realistic scenarios contain hundreds of objects and different agents, with different capabilities, able to perform a large variety of actions. Modeling such worlds soon gets confusing [16]. In other words, not only is it necessary to describe the full semantics of all actions and agents to be able to acquire the goals, but also to represent the dependence relationships among them (without introducing unnecessary orderings). These difficulties are more evident in expressive and temporal planning scenarios, where it becomes necessary to define the temporal annotation of conditions+goals, and even resources. These difficulties pose a challenge and jeopardize the real application of planning. Consequently, there is a growing interest in learning action models because it has the potential to foster the field of AI planning in real-world applications. Particularly, learning in planning provides us with some compelling strategies that allow us to:

- Acquire procedural knowledge through partial observations [5, 15]. Learning from observations of plan traces can be exploited in many scenarios: recognition of past behavior for prediction and anticipation, decision taking and recommendation, programming and modeling, teleoperation, macro recording, sensing and controlling, robotics motion capturing and planning, etc. [4].

- Reduce the human effort in design tasks. Learning is appealing to reduce the expert knowledge and engineering burden that becomes impractical

in some planning scenarios, which are even more complex when temporal features are required. Moreover, in many real-world applications it is expensive, or even impossible, to collect large datasets of observations for training; this happens when the number of samples is limited or when a human needs to perform repetitive actions (learning by demonstration [4]). Therefore, reducing the effort to deal with tractable datasets is desirable [2].

- Improve model accuracy [17]. A wide number of scenarios can benefit from the discovery of a precise model of actions: diagnostic expertise, activity, plan and goal recognition, explainable planning, reformulation and simplification of existing models, etc. [5, 18, 19, 20, 21]. Additionally, a learned model can be used to automatically elaborate similar models for similar scenarios, e.g. knowledge transfer or transfer learning [2].

The underlying idea in learning is to solve the *inverse* problem of planning: rather than creating plans from a given action model, we are now interested in learning an action model from observations of multiple plans. In more detail, the aim of learning action models is to capture common structures (typically conditions, effects and even costs) from an input of observed plans that are consistent with them. By consistent we mean that the learned action model is fully specified and satisfies the constraints imposed by all plans; e.g., if a condition is learned for an action, that condition must hold any time the action appears in any of the plans, which means it must be achieved previously, and analogously for the happening of the effects.

In this paper we propose a Constraint Satisfaction (CS) approach to learn temporal action models, where the model is the representation of the planning domain, from multiple plans that does not require observations on intermediate states. Learning classical models is common [15, 22, 23, 24] but, to the best of our knowledge, learning temporal action models with costs is uncommon. The difficulty of learning temporal models is twofold. First, we need to acquire the action conditions+effects, like in classical model learning. Second, we need to

identify their temporal annotation (in temporal planning conditions/effects can hold/happen at different times) and try to estimate the durations, when they are not precisely given, from input parallel plans where actions can overlap in very different ways [13, 25]. Intuitively, learning temporal models means to learn *which* and *when*. Additionally, we need to estimate the action costs that fit the observations.

This work is inspired by our incipient CS formulation for temporal learning [26]. The work presented here redefines and generalizes our previous formulation in several ways, thus including significant features that are not specifically addressed in other approaches. These features form our main contributions:

1. The CS formulation is inspired by POCL (Partial Order Causal Link [41, 42]) planning, where the causal relationships and the branching schemes are explicitly represented.
2. Reasoning on action duration is more generic now: durations can be fully (un)known or observed with noise.
3. The CS formulation now learns from multiple plans, rather than just from one single plan, which allows us to learn more accurate models and address problems that were previously unmanageable.
4. Reasoning on mutex information is more generic now, as it is formulated over the domain information rather than over the plan information, which leads to a more compact and efficient representation.
5. The CS formulation has been extended to support learning action costs, which is traditionally ignored in literature.
6. We now include a better support for intermediate state observability that allows us to compare to other (classical) learning approaches, thus proving that our approach is highly competitive over IPC (International Planning Competition) domains.

This paper is organized as follows. Section 2 addresses related work. Section 3 presents the planning terminology and formalization, necessary for the CS formulation for learning, detailed in Section 4. In Section 5 we provide a

thorough evaluation of our experiments and compare with other learning approaches. Finally, Section 6 concludes the paper.

## 2. Related Work

From the constraint point of view, constraint acquisition approaches have been studied as user-machine interactive processes to learn constraints from positive and negative examples of solutions. CONACQ [27] is highly related to planning as it learns a constraint network, which represents a sequence of actions, from a set of traces and a library of constraints. ConstraintSeeker [28] and ModelSeeker [29] are also related to constraint acquisition. ConstraintSeeker creates a ranked list of constraints over a constraint catalog to be returned to users. ModelSeeker generalizes models from positive+negative traces and suggests potential solutions. Similarly, QuAcq [30] learns constraints by asking the user to classify partial queries, that is, examples the user needs to classify as solutions or non-solutions. Inductive logic programming [31, 32] constructs first-order clausal theories by also using environment knowledge. In [33], authors use event calculus to learn action models from positive+negative solutions and information about state changes. All these approaches exploit regular structures of constraint problems to find common patterns, but they rely on user's interaction, and focus more on a categorization of constraints than on real learning. Hence, constraint acquisition/induction differ from learning action models as:

- They require a strong interaction and common knowledge, in the form of queries, to communicate between the user and the machine.
- They need positive and negative plans (note that negative plans in planning are useless as they do not achieve the goals).
- The solution found is not a real model of actions, but a set of constraints.

Despite the gap these differences create, constraint acquisition and our CS approach to learn action models share the idea of learning by taking advantage

of constraint reasoning, which means that their learning relies on knowledge-based information, rather than on statistics-based data.

From the planning point of view, learning classical action models is a special type of classification that has been addressed by different approaches that require large datasets of observed plans with information on their intermediate states and actions. Table 1 summarizes these approaches in terms of: i) the technique used; ii) the action model and the supported features; iii) the degree of state/action observability (*full*, if all states/actions in the plans are observed, *vs. partial*, if some states/actions are missing in the plans); and iv) the number of plans used for learning. Note that the last row represents our CS formulation.

ARMS [24] is one of the pioneer approaches, which builds a weighted propositional MAXimum-SATisfiability problem to learn models of actions without costs from scratch. Regarding observability, it requires a partial sequence of states (as some of them can be omitted) and the full sequence of noiseless actions. LAMP [15] supports more expressiveness and learns action models with quantifiers and logical implications by using Markov Logic Networks. It requires the same observability of ARMS. Other approaches like AMAN [34] and RIM [35] include noisy observations and incomplete models, respectively, which means that observations of plans may be captured by imperfect sensors, thus introducing uncertainty under a full degree of observability. LOCM [36, 37] initiated a family of inductive learning systems that use Finite State Machines to learn even from null state information, i.e., without the need of initial, goal or intermediate states. NLOCM [38] is a Numeric extension of LOCM, and it is the only approach in recent literature that learns action costs, with the only observation of the total cost of each plan. LOUGA [39] uses a genetic algorithm to learn an action model with negative conditions, whereas FAMA [22] compiles a planning task and uses a planner to learn from minimal observability, i.e., from incomplete or empty plans with just the initial and goal state. These approaches ignore temporal planning and mutex reasoning, and they typically require hundreds of input plans. Moreover, their statistics-based orientation might lead to learn a little from each plan, but not a model that satisfies every individual plan.

| Name | Technique used | Action model & features supported | Observability on states/actions | Dataset size |
|---|---|---|---|---|
| ARMS | MAX-SAT | STRIPS | Partial states<br>Full actions | 160 plans |
| LAMP | Markov logic networks | STRIPS & quantifiers and logical implications | Partial states<br>Full actions | 100-200 plans |
| AMAN | Probabilistic graphical model | STRIPS & noisy actions | Full states<br>Full actions | 40-200 plans |
| RIM | MAX-SAT | STRIPS & incomplete action models | Full states<br>Full actions | 30-150 plans |
| LOCM | Finite state machines | STRIPS | Partial/null states<br>Full actions | Unknown |
| NLOCM | Finite state automata & constraint programming | Action costs & plan cost | Partial states<br>Full actions | 100-1000 plans |
| LOUGA | Genetic algorithm | STRIPS & negative conditions | Partial states<br>Full actions | 160 plans |
| FAMA | Planning | STRIPS | Partial states<br>Partial actions | 1-10 plans |
| CS | Constraint satisfaction | PDDL2.1 durative actions & noisy durations & action costs & plan cost & mutex info | Partial states<br>Full actions | 1-10 plans |

Table 1: Summary of learning action models approaches (in chronological order). The dataset size for LOCM is not provided in their paper.

This is a limitation when learning from multiple plans. Consequently, exploiting CS technology seems very promising (and worth investigating in comparison to other approaches) in order to address a list of features that are typically ignored. More specifically, in our approach we provide a flexible formulation that integrates PDDL2.1 durative actions with noisy durations, action costs, mutex information representation, partial state observability and full satisfaction of every plan. These are capabilities that altogether are unsupported by other state-of-the-art approaches.

## 3. Terminology and Formalization

### *3.1. Temporal Planning in PDDL2.1*

We define our temporal planning scenario by following the PDDL2.1 structure, i.e., planning domain, problem and plan.

#### *3.1.1. Durative actions in PDDL2.1*

Many planning domains have temporal features that can be expressed as durations associated with actions. Before PDDL2.1, this temporal planning setting was simply ignored (like in PDDL) or addressed in the same conservative way imposed by PDDL: two actions cannot overlap in *any way* if they have conflicting conditions or effects, because conditions must be maintained all over the execution of the durative action and effects only happen when the action ends. This makes it possible to produce reasonable plans in some planning domains, but there exist many practical problems which require a richer model of actions in which better quality plans can be found [13, 40].

PDDL2.1 introduces a rich model of durative actions, which includes local conditions and effects that are temporally annotated (*at start*, *invariant* to be maintained throughout the action execution, and *at end*), to achieve a fuller exploitation of concurrency. In short, a conservative model of temporal planning only has one type of condition and one type of effect, *vs.* the three types of conditions and two types of effects of PDDL2.1. This entails a more precise modeling of the state transitions within the durative interval of the action, which would exclude many valid plans before PDDL2.1. For instance, in PDDL2.1 it is possible to lock a resource *at start* and unlock it *at end*, or to require it just *at start*, rather than maintaining it all over the execution of the action.

*3.1.2. The domain*

Let us assume a hierarchy of types $\mathcal{T}$ and a set of predicates $\mathcal{P}$, with a list of typed parameters over $\mathcal{T}$. A planning domain[1] defines a set of $\mathcal{T}$-parameterized operators $\mathcal{O}$. The number and type of the parameters restrict the subset of predicates to be used per operator. In PDDL2.1, each operator $o \in \mathcal{O}$ is defined by the tuple $\langle \mathsf{dur}(o), \mathsf{cond}_s(o), \mathsf{cond}_i(o), \mathsf{cond}_e(o), \mathsf{eff}_s(o), \mathsf{eff}_e(o), \mathsf{cost}(o) \rangle$. $\mathsf{dur}(o)$ is a positive value indicating the duration of $o$. For simplicity, and without lack of completeness, we assign the duration to the operator (this is typically known as a *simple-time* domain). We will discuss this simplification later in Section 4.2.1. $\mathsf{cond}_s(o), \mathsf{cond}_i(o), \mathsf{cond}_e(o)$, represent the temporally annotated conditions: they must hold before $o$ is executed (*at start*), over all the duration of $o$ (*invariant*) or when $o$ finishes (*at end*), respectively. For simplicity, we assume only positive conditions but negative conditions can be easily managed by creating dummy predicates (e.g., $q = \neg p$). $\mathsf{eff}_s(o)$ and $\mathsf{eff}_e(o)$ represent the two types of effects, which can happen *at start* or *at end* of $o$, and can be positive (asserted) or negative (retracted). We extend the definition of $o$ with a positive $\mathsf{cost}(o)$ that represents the cost of applying $o$, thus allowing planners to optimize the plan cost. A temporal planning domain is defined as $\delta = \langle \mathcal{T}, \mathcal{P}, \mathcal{O} \rangle$.

Figure 1 depicts part of the *zenotravel* domain[2], with its types, predicates and operators[3]. In short, this domain involves transporting people around in planes. It has actions to `board` and `disembark` passengers onto aircraft that can fly between cities at two speeds (fast/slow for `zoom`/`fly` respectively) and consume fuel at different levels according to the speed of travel; fuel can be `refuel`ed. Problem instances require to transport passengers and minimize

[1] Note that in planning, the *domain* refers indistinctly to both the application domain and the domain model. Particularly, the application domain represents the physics of the planning scenario whereas the action model is the representation (in PDDL2.1) to be learned.

[2] See `https://ipc02.icaps-conference.org` for more information on this domain.

[3] PDDL2.1 uses the construct *durative-action* instead of *operator*. To avoid confusion, we distinguish between operators, i.e., templates with parameters, and actions, i.e., instantiated operators where all parameters are grounded to constant values.

some linear combination of time, fuel use and/or cost. The cost is modeled as a particular numeric effect that increases a `cost` expression, but we represent it as $\texttt{cost}(o)$; in our example, $\texttt{cost}(\texttt{board}) = 2$, $\texttt{cost}(\texttt{refuel}) = 5$ and $\texttt{cost}(\texttt{zoom}) = 10$.

*Mutual exclusion (mutex) constraints.* $\mathcal{P}$ has the potential to be large, but this does not mean that any pair of predicates is simultaneously possible in $\delta$. If two predicates $p_i$, $p_j \in \mathcal{P}$ cannot hold together (as this entails a contradiction) we say they are mutex in $\delta$. We define the set of mutex predicates as $\mu(\delta) = \{\langle p_i, p_j \rangle\}$. For instance, in *zenotravel* we know that $\langle$`(at ?x - locatable ?y1 - city)`, `(at ?x - locatable ?y2 - city)`$\rangle$ are mutex if different parameter names involve different values: `?x` cannot be at two different cities `?y1` and `?y2` simultaneously. The domain expert knows this while modeling because it is part of the physics of the domain and can be intuitive, but this knowledge is not explicitly represented in PDDL2.1. Actually, this knowledge is only specified if, given two mutex predicates that appear as effects in the same operator, if one predicate is asserted the other must be retracted and vice versa (contradictory effects). Note that in PDDL2.1 the temporal annotation of mutex effects might vary, i.e., one effect might be asserted as *at start* and the other retracted as *at end*, or vice versa, or both at the same time.

*Static information.* $\mathcal{P}$ may contain predicates that never change in $\delta$, which are known as static. They are part of the physics of the domain but only relevant for the planning grounding stage. Note the `(next ?l1 ?l2 - flevel)` predicate in Figure 1, which represents the natural order between `?l1` and `?l2`. Static predicates are never used as effects since no operator can change the physics of *zenotravel*: one fuel level is always before another. Since static predicates always hold, they could eventually be learned as conditions in all the operators with those parameters. Static information is not explicitly modeled in PDDL2.1, but can be easily detected.

*Input knowledge for learning.* Planners perform some kind of reasoning to discover mutex constraints and static information because it improves the planning

```
(:types locatable city flevel - object       ;description of the object hierarchy
        aircraft person - locatable)
(:predicates (at ?x - locatable ?c - city)   ;plane or person ?x is at city ?c
             (in ?p - person ?a - aircraft)  ;person ?p is onboard plane ?a
             (next ?l1 ?l2 - flevel) ...)    ;fuel-level ?l2 is equivalent to ?l1+1
(:durative-action board                              ;name
 :parameters (?p - person ?a - aircraft ?c - city)  ;three parameters ?p, ?a and ?c
 :duration (= ?duration 20)                          ;duration
 :condition (and (at start (at ?p ?c))               ;person ?p initially at ?c
                 (over all (at ?a ?c)))              ;plane ?a must remain at ?c
 :effect (and (at start (not (at ?p ?c)))            ;initially person ?p is no longer at ?c
              (at end (in ?p ?a))                    ;finally person ?p is onboard ?a
              (at end (increase (cost) 2))))         ;finally increasing the cost
(:durative-action refuel
 :parameters (?a - aircraft ?c - city ?l1 ?l2 - flevel)
 :duration (= ?duration 73)
 :condition (and (at start (fuel-level ?a ?l1))  ;initial fuel-level ?l1 for plane ?a
                 (at start (next ?l1 ?l2))       ;ordering between ?l1 and ?l2
                 (over all (at ?a ?c)))
 :effect (and (at end (fuel-level ?a ?l2))       ;new fuel-level ?l2 for plane ?a
              (at end (not (fuel-level ?a ?l1))) ;removing previous fuel-level ?l1 for plane ?a
              (at end (increase (cost) 5))))
(:durative-action zoom
 :parameters (?a - aircraft ?c1 ?c2 - city ?l1 ?l2 ?l3 - flevel)
 :duration (= ?duration 100)
 :condition (and (at start (at ?a ?c1)) (at start (fuel-level ?a ?l1))
                 (at start (next ?l2 ?l1)) (at start (next ?l3 ?l2)))
 :effect (and (at start (not (at ?a ?c1))) (at end (at ?a ?c2))  ;moving plane ?a from city ?c1 to ?c2
              (at end (not (fuel-level ?a ?l1))) (at end (fuel-level ?a ?l3)) ;fuel consumption, from ?l1 to ?l3
              (at end (increase (cost) 10))))
```

Figure 1: Fragment of the *zenotravel* domain with comments on the meaning. Other operators are `debark` and `fly`. This is known as the reference or *ground truth* model.

task. Similarly, a learning task can also be improved by providing additional input knowledge. First, giving the set of mutex predicates $\mu(\delta)$. Second, giving a set of predicates $\mathcal{P}' \subseteq \mathcal{P}$, where $\mathcal{P}'$ contains no static predicates. This knowledge is very valuable for the learning task, as we will see in Section 3.3.

### *3.1.3. The problem*

Let us assume a domain $\delta = \langle \mathcal{T}, \mathcal{P}, \mathcal{O} \rangle$. Given a set of $\mathcal{T}$-typed constant values, we define $\mathcal{V}$ and $\mathcal{A}$. $\mathcal{V}$ is the set of Boolean variables instantiating these values in $\mathcal{P}$, thus defining a mapping between $\mathcal{P}$ and $\mathcal{V}$. $\mathcal{A}$ is the set of durative actions, which are instantiated from the operators $\mathcal{O}$. All the parameters in $\mathcal{A}$ are grounded, and the same for their conditions+effects, which are thereby instantiated and mapped in $\mathcal{V}$. For instance, from Figure 1 and given the constant values {`plane1 - aircraft`, `person1 - person`, `city0 - city`} we obtain the variables {`(at plane1 city0)`, `(at person1 city0)`, `(in person1 plane1)`} and the action `(board person1 plane1 city0)`. We also define a state $\mathcal{S}$ as an assignment of true/false values to variables in $\mathcal{V}$. $\mathcal{S}$ is a full state if $|\mathcal{S}| = |\mathcal{V}|$ and a partial state if $|\mathcal{S}| < |\mathcal{V}|$. A temporal planning problem $\rho$ for $\delta$ is defined as $\rho = \langle \delta, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$, where $\mathcal{I}$ is the initial full state and $\mathcal{G}$ is the goal state. Although $\mathcal{G}$ can be a partial or full state, it is typically defined as a partial state.

### *3.1.4. The plan*

Let us assume a problem $\rho = \langle \delta, \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$. A temporal plan for $\rho$ is a tuple $\pi(\rho) = \langle \{\langle t_1, a_1 \rangle, \langle t_2, a_2 \rangle \dots \langle t_n, a_n \rangle\}, \mathcal{K} \rangle$. Each $\langle t_i, a_i \rangle$ contains an action $a_i \in \mathcal{A}$, where its duration can be (un)known, and $t_i$ as the start time of $a_i$. Actions in $\mathcal{A}$ can have several occurrences in $\pi(\rho)$ provided they start at different times, and different actions can start at the same time because the plan is a parallel one. $\mathcal{K}$ is the cost of the plan. $\pi(\rho)$ induces a chronologically-ordered sequence of full states $\langle \mathcal{S}_0 \dots \mathcal{S}_{end} \rangle$, where $\mathcal{S}_0 = \mathcal{I}$ and $\mathcal{G} \subseteq \mathcal{S}_{end}$. The term *makespan* represents the time of the state $\mathcal{S}_{end}$. Figure 2 shows a fragment of a PDDL2.1 plan for *zenotravel* that includes the start time, the action and its

```
1: (refuel plane1 city0 fl1 fl2) [73]
1: (board person2 plane2 city1) [20]
1: (board person1 plane1 city0) [20]
75: (zoom plane1 city0 city1 fl2 fl1 fl0) [100]
22: (zoom plane2 city1 city2 fl3 fl2 fl1) [100]
176: (debark person1 plane1 city1) [30]
...
```

Figure 2: Fragment of a temporal plan for *zenotravel* with a cost $\mathcal{K}$.

duration.

### *3.2. Constraint Satisfaction Problem*

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

An evaluation of values to variables is consistent if it does not violate any of the constraints in $\mathcal{C}$. A consistent evaluation is a solution if it includes values for all variables in $\mathcal{X}$. A CSP can have many solutions. If we do not define a metric over $\mathcal{X}$, we are dealing with a pure satisfaction problem, rather than a constraint optimization problem. Thus, many solutions are possible and equally valid.

### *3.3. Learning Task*

Let us consider a domain $\delta = \langle \mathcal{T}, \mathcal{P}, \mathcal{O} \rangle$ and a set of empty operators $\mathcal{O}?$. $\mathcal{O}$ and $\mathcal{O}?$ are equal, but by empty we mean that the conditions+effects (and their temporal annotation *at start*, *over all* or *at end*), durations and costs are unknown. We assume the name, or a unique identifier, and the parameters of operators in $\mathcal{O}?$ are known. Such assumption, frequent in literature of learning action models, may seem a bit strong but it is minimally necessary to identify relationships and establish common structures. The parameters are needed to automat-

ically generate a set of candidate predicates that can be potentially used in every operator. For instance, `(board ?p - person ?a - aircraft ?c - city)` and `(refuel ?a - aircraft ?c - city ?l1 ?l2 - flevel)` are empty operators for the operators of Figure 1. Let us also consider a set of $n$ problems for $\delta$ and their corresponding observed plans $\Pi_n(\delta) = \{\pi(\rho_1) \dots \pi(\rho_n)\}$. We define the learning task from $\Pi_n(\delta)$ as the tuple $\mathcal{L}_n = \langle \delta, \Pi_n(\delta), \mu(\delta), \mathcal{O}? \rangle$, where the mutex information in $\mu(\delta)$ can be optionally empty.

A solution for the learning task $\mathcal{L}_n$, denoted as $\mathcal{O}(\mathcal{L}_n)$, is an approximation of $\mathcal{O}?$ to the original operators in $\mathcal{O}$, used as the reference model. It is an approximation because the learned operators are not always identical to the reference ones; e.g., the number and the temporal annotation of conditions/effects and the durations/costs may differ. $\mathcal{O}(\mathcal{L}_n)$ must satisfy all plans in $\Pi_n(\delta)$ (completeness) and the operator model must imply no contradictions nor mutexes in any plan (soundness). Broadly speaking, the learning task needs to complete the operators in $\mathcal{O}?$ by reasoning over a collection of plans. This reasoning is a little inductive and a little abductive: inductive because it makes inferences based on what is observed on the plans and abductive because it forms a probable, though not necessarily unique, conclusion from those plans. More formally, $\mathcal{O}(\mathcal{L}_n)$ must be consistent with the information in $\mathcal{L}_n$. This means that if actions $\mathcal{A}_i$ in every $\pi(\rho_i)$ are instantiated versions of the operators in $\mathcal{O}(\mathcal{L}_n)$, all plans in $\Pi_n(\delta)$ are consistent: i) $\mathcal{G}_i$ is achieved from $\mathcal{I}_i$; and ii) the learned durations/costs induce the observed makespan/cost of $\pi(\rho_i)$.

## 4. CS Formulation for Learning Temporal Action Models

Our CS formulation represents a learning task $\mathcal{L}_n$, thus encoding the conditions and effects (i.e., causal relationships), duration and cost of each operator, and the actions of the observed plans. Such formulation follows a POCL planning fashion, 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.

### *4.1. Preprocessing*

Before proceeding with the formulation, we need to populate the potential candidate predicates that can be learned as conditions+effects.

Given the domain $\delta$ and the set $\mathcal{O}?$, we define the alphabet of $o \in \mathcal{O}?$, denoted as $\alpha(o)$, as the set of all predicates $p \in \mathcal{P}$ whose types belong to the types of the parameters of $o$. For instance, according to Figure 1, $\alpha$(`board`)={`(at ?p - person ?c - city)`, `(at ?a - aircraft ?c - city)`, `(in ?p - person ?a - aircraft)`}. This alphabet produces the candidate predicates: the sets $\{p_i\}$ and $\{p_i \cup \neg p_i\}$, where $p_i \in \alpha(o)$, denote the candidates that can be conditions and effects of $o$, respectively. We deal with positive conditions and positive+negative effects, as the same predicate can be asserted and retracted at different times. The number of potential candidates is, therefore, $3^*|\alpha(o)|$.

In order to facilitate the formulation, we create a mapping between the operators in the domain and the actions in a plan. For each $\pi \in \Pi_n(\delta)$, let $o_\pi$ be an action in $\pi$ that is the result of instantiating an operator $o \in \mathcal{O}?$. Let $p \in \alpha(o)$ be a predicate of $o$ that is instantiated for any action $o_\pi$, and denoted as $p_\pi$. For instance, given the operator $o$=`(board ?p - person ?a - aircraft ?c - city)` and the predicate $p$=`(at ?p - person ?c - city)`, a possible mapping is for action $o_\pi$=`(board person1 plane1 city0)` and the variable $p_\pi$=`(at person1 city0)`. The learning task implies that, for example, if $p$ is learned as an *over all* condition in $o$, any $p_\pi$ must be consistent with such decision in any $\pi$.

### *4.2. The Formulation*

#### *4.2.1. The variables*

We define nine types of variables (X1..X9), which are grouped into two blocks that are shown in Figure 3. X1..X4 model the information of the domain ($o \in \mathcal{O}?$ and $p \in \alpha(o)$). X5..X9 need to be repeated per each mapped action/variable ($o_\pi/p_\pi$) present in each plan $\pi \in \Pi_n(\delta)$. In comparison with our previous

| Id. | Variable | Domain |
|---|---|---|
| X1 | $\mathsf{dur}(o)$ | $\mathbb{Z}^+ \cup \{0\}$ |
| X2 | $\mathsf{cost}(o)$ | $\mathbb{Z}^+ \cup \{0\}$ |
| X3 | $\mathsf{cond}(p, o)$ | $\{$*no_cond, at_start, overall, at_end*$\}$ |
| X4.1 | $\mathsf{eff}(p, o)$ | $\{$*no_eff, at_start, at_end*$\}$ |
| X4.2 | $\mathsf{eff}(\neg p, o)$ | $\{$*no_eff, at_start, at_end*$\}$ |
| X5 | $\mathsf{start}(o_\pi)$ | $[0..$*makespan*$(\pi)]$ |
| X6 | $\mathsf{end}(o_\pi)$ | $\mathsf{end}(o_\pi) = \mathsf{start}(o_\pi) + \mathsf{dur}(o)$ |
| X7.1 | $\mathsf{req\_start}(p_\pi, o_\pi)$ | $\{\mathsf{start}(o_\pi), \mathsf{end}(o_\pi)\}$ |
| X7.2 | $\mathsf{req\_end}(p_\pi, o_\pi)$ | $\{\mathsf{start}(o_\pi), \mathsf{end}(o_\pi)\}$ |
| X8.1 | $\mathsf{time}(p_\pi, o_\pi)$ | $\{\mathsf{start}(o_\pi), \mathsf{end}(o_\pi)\}$ |
| X8.2 | $\mathsf{time}(\neg p_\pi, o_\pi)$ | $\{\mathsf{start}(o_\pi), \mathsf{end}(o_\pi)\}$ |
| X9 | $\mathsf{sup}(p_\pi, o_\pi)$ | $\emptyset \cup \{o'_\pi\} \in \pi$ |

Figure 3: Formulation of variables and their domain.

formulation in [26], all variables have been now redefined to differentiate between operators and actions of multiple input plans, and X2..X4 are completely new.

X1 models the duration of the operator and X2 its cost (for simplicity, we use discrete domains). As indicated in Section 3.1.2, we assign durations and costs to operators. Although we might assign different durations and costs to different actions (thus being parameter-dependent) by creating more variables, distinct problems over the same domain are unrelated in PDDL2.1. For instance, the value `plane1` might represent a different plane in each problem in *zenotravel*, which means that learning the duration/cost for different `board` or `refuel` actions over `plane1` would depend on every particular problem, not on the domain. This is why we restrict durations and costs to operators.

X3 represents if $p$ is a condition for $o$. The possible values are *no_cond* ($p$ is not learned as a condition), *at_start*, *overall* or *at_end*, meaning that $p$ is annotated as an *at start*, *invariant* or *at end* condition, respectively. X4.1 (X4.2) represent if $p$ ($\neg p$) is learned as a positive (negative) effect. The possible values are *no_eff* (if it is not an effect), *at_start* or *at_end*. Note that $\neg p$ could happen

*at_start* and $p$ *at_end*, which is a typical way to model resource usage in temporal planning. Also note that we allow $p$ to appear both as a condition and as an effect in the same operator, but, for simplicity, we restrict each condition/effect to appear at most once in every operator. For instance, although in PDDL2.1 $p$ could theoretically appear both as *at_start* and *at_end* conditions in the same operator, in practical domains of IPC this does not happen. In other words, a temporally simple subset of potential PDDL2.1, where deep levels of concurrency [25] are not exploited, is used in the planning competitions. If the same condition/effect $p$ had to appear more than once in the same operator, we could simply extend our formulation to create new versions $\{p_1, p_2 \dots p_n\}$, of $p$ and make all $p_i$ have the same value.

X5 (X6) represent the start (end) time of $o_\pi$. X7.1 and X7.2 define the interval $[\mathsf{req\_start}(p_\pi, o_\pi)..\mathsf{req\_end}(p_\pi, o_\pi)]$ throughout $p_\pi$ must hold for $o_\pi$ (provided the mapped $\mathsf{cond}(p, o) \neq$*no_cond*). This interval allows us to model the three types of conditions of X3. X8.1 models the time when $p_\pi$ happens in $o_\pi$, provided $\mathsf{eff}(p, o) \neq$*no_eff*. X8.2 is analogous, but for $\neg p_\pi$. X9 represents a causal link relationship like in POCL planning, representing that action $o'_\pi$ supports $p_\pi$, which is required by $o_\pi$. If $\mathsf{cond}(p, o)$=*no_cond*, meaning that $p$ is not a condition of $o$ (and, consequently, $p_\pi$ is not a condition of $o_\pi$), then $\mathsf{sup}(p_\pi, o_\pi) = \emptyset$, i.e., the empty supporter. We do not include the variable $\mathsf{sup}(\neg p_\pi, o_\pi)$ because we only deal with positive conditions and, therefore, $\neg p_\pi$ is never required as a condition.

Additionally, we create two dummy actions (and their respective dummy operators) per problem $\rho \in \{\rho_1 \dots \rho_n\}$. First, $\mathsf{init}_\rho$ represents $\mathcal{I}$ ($\mathsf{dur}(\mathsf{init}_\rho) = \mathsf{cost}(\mathsf{init}_\rho) = \mathsf{start}(\mathsf{init}_\rho) = 0$). $\mathsf{init}_\rho$ has no cond, req_start, req_end and sup variables because it has no conditions. $\mathsf{init}_\rho$ has as many $\mathsf{eff}(p_i, \mathsf{init}_\rho)$=*at_end* as $p_i \in \mathcal{I}$, and analogously for the false variables $\neg p_j \in \mathcal{I}$, as $\mathcal{I}$ is a full state. Second, $\mathsf{goal}_\rho$ represents $\mathcal{G}$ ($\mathsf{dur}(\mathsf{goal}_\rho) = \mathsf{cost}(\mathsf{goal}_\rho) = 0$ and $\mathsf{start}(\mathsf{goal}_\rho)$=*makespan(π)*). $\mathsf{goal}_\rho$ has as many $\mathsf{cond}(p_i, \mathsf{goal}_\rho)$=*at_start* as positive conditions $p_i \in \mathcal{G}$. $\mathsf{goal}_\rho$ has no eff and time variables because it has no effects.

Including observations of intermediate states is straightforward. We simply

define a dummy action+operator $\mathsf{obs}(p,t)$, analogous to $\mathsf{goal}$, where $p$ is the observed predicate at time $t$ ($\mathsf{cond}(p,\mathsf{obs}(p,t))$=*at_start*, $\mathsf{sup}(p,\mathsf{obs}(p,t)) \neq \emptyset$, and no $\mathsf{eff}$+$\mathsf{time}$ variables at all).

*4.2.2. The constraints*

We formulate all the necessary constraints that allow us to learn the conditions, effects and costs. The fact of modeling constraints in terms of POCL planning, explicitly representing the causal links and threat resolution, allows us to better capture the insights of the observed plans and, consequently, of the action model. Figure 4 shows the two blocks of constraints. Clearly, the operator variables impose constraints that must be consistent with the actions and vice versa. C1..C12 model the planning constraints and C13 the mutex constraints if $\mu(\delta)$ is not empty in $\mathcal{L}_n$. In comparison with the formulation in [26], all constraints are now redefined to represent the operator-action duality, and C12..C13 are completely new.

C1..C4 represent the constraints for $\mathsf{cond}$. C1 guarantees that if $p$ is not learned as a condition in $o$, it has no support in its instantiated actions in the plan, and vice versa (note that this is an *if and only if* constraint). C2..C4 enforce the *at start*, *invariant* and *at end* conditions in terms of the X7 variables. C5..C7 represent the constraints for $\mathsf{eff}$. In C5, if $p$ is not learned as an effect in $o$ then no instantiated actions from $o$ can be supporters of the mapped $p_\pi$. C6.1 (C6.2) enforce the *at start* positive (negative) effects in terms of the X8 variables; analogously, C7.1 (C7.2) for *at end* positive (negative) effects. C8 prevents two operators (possibly equal) from having contradictory effects at the same time. C9 enforces that if the same operator $o$ requires and deletes $p$, the effect cannot happen before the condition. Note the "$\geq$" inequality: if one condition and one effect of the same operator appear at the same time, PDDL2.1 considers the condition is checked instantly before the effect. C10 models the causal link $\langle o'_\pi, p_\pi, o_\pi \rangle$, which means that $p_\pi$ must happen before it is required; intuitively, $p_\pi$ must be supported by $o'_\pi$ before being used by $o_\pi$. Note the "$<$" constraint instead of "$\leq$". PDDL2.1 assumes an $\epsilon > 0$, where $\epsilon$ is a small tolerance that

| Id. | Constraint |
|---|---|
| C1 | **iff** $(\mathsf{cond}(p,o)=no\_cond)$ **then** $\mathsf{sup}(p_\pi,o_\pi)=\emptyset$ |
| C2 | **if** $(\mathsf{cond}(p,o)=at\_start)$ **then** $\mathsf{req\_start}(p_\pi,o_\pi)=\mathsf{req\_end}(p_\pi,o_\pi)=\mathsf{start}(o_\pi)$ |
| C3 | **if** $(\mathsf{cond}(p,o)=overall)$ **then** $(\mathsf{req\_start}(p_\pi,o_\pi)=\mathsf{start}(o_\pi))$ **and** $(\mathsf{req\_end}(p_\pi,o_\pi)=\mathsf{end}(o_\pi))$ |
| C4 | **if** $(\mathsf{cond}(p,o)=at\_end)$ **then** $\mathsf{req\_start}(p_\pi,o_\pi)=\mathsf{req\_end}(p_\pi,o_\pi)=\mathsf{end}(o_\pi)$ |
| C5 | **if** $(\mathsf{eff}(p,o)=no\_eff)$ **then forall** $o'_\pi$ that requires $p_\pi$: $\mathsf{sup}(p_\pi,o'_\pi)\neq o_\pi$ |
| C6.1 | **if** $(\mathsf{eff}(p,o)=at\_start)$ **then** $\mathsf{time}(p_\pi,o_\pi)=\mathsf{start}(o_\pi)$ |
| C6.2 | **if** $(\mathsf{eff}(\neg p,o)=at\_start)$ **then** $\mathsf{time}(\neg p_\pi,o_\pi)=\mathsf{start}(o_\pi)$ |
| C7.1 | **if** $(\mathsf{eff}(p,o)=at\_end)$ **then** $\mathsf{time}(p_\pi,o_\pi)=\mathsf{end}(o_\pi)$ |
| C7.2 | **if** $(\mathsf{eff}(\neg p,o)=at\_end)$ **then** $\mathsf{time}(\neg p_\pi,o_\pi)=\mathsf{end}(o_\pi)$ |
| C8 | **if** $(\mathsf{eff}(p,o)\neq no\_eff)$ **and** $(\mathsf{eff}(\neg p,o')\neq no\_eff)$ **then** $\mathsf{time}(p_\pi,o_\pi)\neq\mathsf{time}(\neg p_\pi,o'_\pi)$ |
| C9 | **if** $(\mathsf{cond}(p,o)\neq no\_cond)$ **and** $(\mathsf{eff}(\neg p,o)\neq no\_eff)$ **then** $\mathsf{time}(\neg p_\pi,o_\pi)\geq\mathsf{req\_end}(p_\pi,o_\pi)$ |
| C10 | **if** $(\mathsf{sup}(p_\pi,o_\pi)=o'_\pi)$ **then** $\mathsf{time}(p_\pi,o'_\pi)<\mathsf{req\_start}(p_\pi,o_\pi)$ |
| C11 | **if** $(\mathsf{sup}(p_\pi,o_\pi)=o'_\pi)$ **and** $(\mathsf{eff}(\neg p,o^{\neg p})\neq no\_eff)$ **and** $(o_\pi\neq o_\pi^{\neg p})$ **then** $(\mathsf{time}(\neg p_\pi,o_\pi^{\neg p})<\mathsf{time}(p_\pi,o'_\pi))$ **or** $(\mathsf{time}(\neg p_\pi,o_\pi^{\neg p})>\mathsf{req\_end}(p_\pi,o_\pi))$ |
| C12 | $\sum occurrences(o_\pi)*\mathsf{cost}(o)=\mathcal{K}_\pi$ |
| C13 | $\forall\ \langle p_i,p_j\rangle\in\mu(\delta)$ to be used in $o$: |
| C13.1 | **if** $(\mathsf{cond}(p_i,o)\neq no\_cond)$ **and** $(\mathsf{cond}(p_j,o)\neq no\_cond)$ **then** $\mathsf{cond}(p_i,o)\neq\mathsf{cond}(p_j,o)$ |
| C13.2 | **if** $(\mathsf{eff}(p_i,o)\neq no\_eff)$ **and** $(\mathsf{eff}(p_j,o)\neq no\_eff)$ **then** $\mathsf{eff}(p_i,o)\neq\mathsf{eff}(p_j,o)$ |
| C13.3 | **if** $(\mathsf{eff}(p_i,o)\neq no\_eff)$ **then** $\mathsf{eff}(\neg p_j,o)\neq no\_eff$ |
| C13.4 | **if** $(\mathsf{eff}(p_j,o)\neq no\_eff)$ **then** $\mathsf{eff}(\neg p_i,o)\neq no\_eff$ |

Figure 4: Formulation of planning and mutex constraints. For simplicity, the value of irrelevant variables is not bounded here. For instance, if $\mathsf{cond}(p,o)=no\_cond$, both $\mathsf{req\_start}(p_\pi,o_\pi)$ and $\mathsf{req\_end}(p_\pi,o_\pi)$ become irrelevant.

avoids intersection between the time when effects are supported and when they are required. When time is modeled in $\mathbb{Z}^+$, $\epsilon = 1$ and "$\leq$" becomes "$<$". Given a causal link $\langle o'_\pi, p_\pi, o_\pi \rangle$, C11 avoids the possible threat of action $o_\pi^{\neg p}$ with its effect $\neg p_\pi$; intuitively, $o_\pi^{\neg p}$ could break such a causal link. The two ways to solve a threat in POCL, which impose a possible ordering of $o_\pi^{\neg p}$, are modeled in C11: promotion, where $o_\pi^{\neg p}$ happens before $p_\pi$ is supported, or demotion, where $o_\pi^{\neg p}$ happens after $p_\pi$ is required. C12 encodes the linear constraint for the cost $\mathcal{K}_\pi$ of plan $\pi$, where $occurrences(o_\pi)$ is the number of instances of $o_\pi$ in $\pi$. Finally, C13 represents the mutex constraints. If two mutex predicates are learned as conditions (effects) in an operator, they cannot happen at the same time, which is enforced by C13.1 (C13.2). C13.3 guarantees that if $p_i$ is learned as an effect in $o$, $\neg p_j$ must also be an effect, whereas C13.4 guarantees the opposite constraint. These two last constraints are essential to learn negative effects[4]. In absence of them, and without negative conditions nor intermediate state observability, the learning task will never learn negative effects (e.g., in *zenotravel* a plane could be learned to be in two cities simultaneously). It is important to note that, a priori, we cannot foresee the temporal annotation (X8 variable) for mutex predicates, as they happen indistinctly at the same or different times depending on the domain. For instance, if we consider the mutex $\langle$`(at ?x - person ?y1 - city)`, `(in ?x - person ?y2 - aircraft)`$\rangle$ in `board`, the first predicate is retracted *at start* and the second one is asserted *at end*. However, if we consider the mutex $\langle$`(fuel-level ?a - aircraft ?y1 - flevel)`, `(fuel-level ?a - aircraft ?y2 - flevel)`$\rangle$ in `refuel` and `zoom`, the predicates are asserted and retracted *at end*, but they could perfectly be annotated as *at start* and *at end*, respectively. This different way of shaping the semantics for mutex, even within the same domain, reveals the difficulty of the modeling task even for an expert

[4]State-of-the-art learning approaches do not reason on mutex and they learn negative effects by requiring intermediate states and/or a full goal state. We avoid this requirement by performing an automated reasoning on mutex that allows us to successfully infer negative effects.

engineer.

*4.2.3. A working example*

Let us consider the operator `board` of Figure 1, with $\alpha$(`board`)={`(at ?p - person ?c - city)`, `(at ?a - aircraft ?c - city)`, `(in ?p - person ?a - aircraft)`}. Let us also consider the action `(board person2 plane2 city1)` of the plan in Figure 2. Following the definition of variables of Figure 3, the resulting variables are shown in Figure 5. The constraints for this operator and action are created following Figure 4. Figure 6 shows only the results for C1 and C2 constraints, but the remaining constraints are created analogously.

*4.2.4. Formal properties*

The formulation presents interesting properties: soundness and completeness (inherited from POCL planning), and polynomial size.

**Lemma 1.** *Soundness. The formulation is sound.*

**Proof sketch.** By soundness of the formulation we refer to the property that it allows us to find a PDDL2.1 sound plan that satisfies all the given constraints of the model. A plan $\pi$ is sound *w.r.t.* a model of actions if all conditions of any action, including the dummy actions, are satisfied at their temporal annotation and no contradictions appear. The formulation provides the same twofold sound branching scheme defined by POCL planning. First, conditions are satisfied thanks to C10, which guarantees that all conditions are supported before being used, and no threats break those causal links thanks to the ordering imposed by C11. Second, no contradictions appear thanks to C8, which avoids contradictory effects, and C13, which guarantee that no mutexes happen in the plan. Hence, $\pi$ is sound and, consequently, the formulation is also sound. □

**Lemma 2.** *Completeness. Any possible solution $\mathcal{O}(\mathcal{L}_n)$ is computable by solving the given formulation.*

**Proof sketch.** The formulation creates a CSP that encodes all constraints for $\mathcal{L}_n$ and also checks the POCL plan validity conditions (as shown in Lemma 1). Let us suppose, by contradiction, that a possible solution $\mathcal{O}(\mathcal{L}_n)'$ is not found.

dur(board)

cost(board)

cond((at ?p - person ?c - city), board)

cond((at ?a - aircraft ?c - city), board)

cond((in ?p - person ?a - aircraft), board)

eff((at ?p - person ?c - city), board)

eff((at ?a - aircraft ?c - city), board)

eff((in ?p - person ?a - aircraft), board)

eff(not-(at ?p - person ?c - city), board)

eff(not-(at ?a - aircraft ?c - city), board)

eff(not-(in ?p - person ?a - aircraft), board)

start((board person2 plane2 city1))

end((board person2 plane2 city1))

req_start((at person2 city1), (board person2 plane2 city1))

req_start((at plane2 city1), (board person2 plane2 city1))

req_start((in person2 plane2), (board person2 plane2 city1))

req_end((at person2 city1), (board person2 plane2 city1))

req_end((at plane2 city1), (board person2 plane2 city1))

req_end((in person2 plane2), (board person2 plane2 city1))

time((at person2 city1), (board person2 plane2 city1))

time((at plane2 city1), (board person2 plane2 city1))

time((in person2 plane2), (board person2 plane2 city1))

time(not-(at person2 city1), (board person2 plane2 city1))

time(not-(at plane2 city1), (board person2 plane2 city1))

time(not-(in person2 plane2), (board person2 plane2 city1))

sup((at person2 city1), (board person2 plane2 city1))

sup((at plane2 city1), (board person2 plane2 city1))

sup((in person2 plane2), (board person2 plane2 city1))

Figure 5: A simple example of CS variables for `board` and `(board person2 plane2 city1)`.

**iff** (cond((at ?p - person ?c - city), board)=*no_cond*) **then**
sup((at person2 city1), (board person2 plane2 city1)) = ∅
**iff** (cond((at ?a - aircraft ?c - city), board)=*no_cond*) **then**
sup((at plane2 city1), (board person2 plane2 city1)) = ∅
**iff** (cond((in ?p - person ?a - aircraft), board)=*no_cond*) **then**
sup((in person2 plane2), (board person2 plane2 city1)) = ∅
**if** (cond((at ?p - person ?c - city), board)=*at_start*) **then**
req_start((at person2 city1), (board person2 plane2 city1)) =
req_end((at person2 city1), (board person2 plane2 city1)) =
start((board person2 plane2 city1))
**if** (cond((at ?a - aircraft ?c - city), board)=*at_start*) **then**
req_start((at plane2 city1), (board person2 plane2 city1)) =
req_end((at plane2 city1), (board person2 plane2 city1)) =
start((board person2 plane2 city1))
**if** (cond((in ?p - person ?a - aircraft), board)=*at_start*) **then**
req_start((in person2 plane2), (board person2 plane2 city1)) =
req_end((in person2 plane2), (board person2 plane2 city1)) =
start((board person2 plane2 city1))

Figure 6: A simple example of CS constraints for `board` and `(board person2 plane2 city1)`. For simplicity, only C1 and C2 constraints of Figure 4 are shown.

Since we perform a complete exploration of consistent values for all variables in the CSP, that is only possible if $\mathcal{O}(\mathcal{L}_n)'$ represents an evaluation of values to variables that is inconsistent because it violates any of the C1..C13 constraints. Violation of any constraint means that $\mathcal{O}(\mathcal{L}_n)'$ cannot be a solution for $\mathcal{L}_n$. Consequently, if a solution exists it will be eventually found, thus guaranteeing completeness. □

**Lemma 3.** *The number of variables and constraints of the formulation is polynomial in the size of the alphabets, and number of instantiated predicates and actions in the plans.*

**Proof sketch.** Given a learning task $\mathcal{L}_n$, the generation of variables and constraints in the formulation is a finite process because the number of operators, predicates and actions in the plans are finite. Let $U_\alpha$ be the upper bound on the size of the alphabets, and $U_\pi$ the upper bound on the number of actions in each of the $n$ input plans. The number of variables is bounded by $\mathrm{O}(U_\alpha{\cdot}n{\cdot}U_\pi)$ in X8; intuitively, X8 represents the happening times for the potential positive+negative effects ($2{*}U_\alpha$) in the actions ($U_\pi$) on any of the $n$ plans. On the other hand, the number of constraints is bounded by $\mathrm{O}(U_\alpha{\cdot}n{\cdot}U_\pi^3)$ in C11; intuitively, C11 solves the possible threats that might appear for a potential condition ($U_\alpha$) and involve any combination of three actions ($o_\pi, o'_\pi, o_\pi^{\neg p}$, i.e., $U_\pi^3$) on any of the $n$ plans. Therefore, the formulation size is polynomial. □

### *4.3. Solving the Formulation*

#### *4.3.1. Use of a CS-based solver*

CS, CSP and SATisfiability are highly related frameworks [43]. Constraint formulations can be mapped into SAT ones and vice versa. However, non-boolean variables like ours make the SAT encodings more complex, e.g., we need specific clauses to ensure that a SAT variable is given one and only one value [43]. Despite modern SAT solvers are very efficient, we are mainly interested in a CS formulation to represent $\mathcal{L}_n$ that easily encodes the constraints of multiple plans, so we opt for a CS-based solver. Concretely, we have used `Choco` (`www.choco-solver.org`), an open-source Java library that provides an

object-oriented API for solving CSPs.

For solving the formulation in Choco, we deal with a pure satisfaction problem. Although a metric allows the solver to prioritize over the space of solutions, we have not found a conclusive metric to decide the best learning. We have investigated different metrics, e.g., maximizing the use of conditions and/or causal links, minimizing the number of unused side effects, etc., but one learned model (i.e., solution) is not better because it has more or less conditions or causal links, specially in presence of static information. Intuitively, a metric cannot substitute the unknown intention of the domain expert. Additionally, the use of a metric has not a definitive impact in reducing the variance of the learned models, as this depends on the solver and machine performance.

*4.3.2. A Choco example*

Using Choco is simple. First, we need to create a model and declare its variables. This is shown in Figure 7, with only the X1-X6 variables, for the example of Figure 5. The method `intVar` requires the name of the variable and the min/max value of its domain. We restrict the values of dur and cost to large values, i.e., $MAX\_DURATION$=$MAX\_COST$=1000. The cond and eff variables are restricted to 3 (*no_cond=0, at_start=1, at_end=2, overall=3*) and 2 (*no_eff=0, at_start=1, at_end*=2), respectively. The remaining variables are declared analogously.

Next, we need to build the constraints on such variables, as shown in Figure 8. Choco provides many arithmetic and logic expressions to be posted in the model. For simplicity, in the Figure we only show the constraints for the X6 variable and C1, as presented in Figure 6, but the others are similar.

Finally, we need to launch the resolution process by executing `m.getSolver().solve()`, which computes a feasible solution. Figure 9 shows part of a solution, i.e., a learned model for the operators `board`, `refuel` and `zoom`. In comparison with the original operators, considered as the reference or ground truth model, given in Figure 1, we can see that learning the exact duration and cost is difficult, as some learned values are incorrect. On the other hand, most conditions/effects

```
Model m = new Model();
m.intVar("dur(board)", 0, MAX_DURATION);
m.intVar("cost(board)", 0, MAX_COST);
m.intVar("cond((at ?p - person ?c - city),board)", 0, 3);
m.intVar("cond((at ?a - aircraft ?c - city),board)", 0, 3);
m.intVar("cond((in ?p - person ?a - aircraft),board)", 0, 3);
m.intVar("eff((at ?p - person ?c - city),board)", 0, 2);
m.intVar("eff((at ?a - aircraft ?c - city),board)", 0, 2);
m.intVar("eff((in ?p - person ?a - aircraft),board)", 0, 2);
m.intVar("eff(not-(at ?p - person ?c - city),board)", 0, 2);
m.intVar("eff(not-(at ?p - person ?c - city),board)", 0, 2);
m.intVar("eff(not-(at ?p - person ?c - city),board)", 0, 2);
m.intVar("start((board person2 plane2 city1))", 0, makespan);
m.intVar("end((board person2 plane2 city1))", 0, makespan);
```

Figure 7: A simple example in Choco for X1-X6 variables of Figure 5.

```
m.arithm("end((board person2 plane2 city1))", "=",
  "start((board person2 plane2 city1))", "+", "dur(board)");
m.ifOnlyIf(m.arithm("cond((at ?p - person ?c - city),board)", "=", 0),
  m.arithm("sup((at person2 city1),(board person2 plane2 city1))", "=", 0));
m.ifOnlyIf(m.arithm("cond((at ?a - aircraft ?c - city),board)", "=", 0),
  m.arithm("sup((at plane2 city1),(board person2 plane2 city1))", "=", 0));
m.ifOnlyIf(m.arithm("cond((in ?p - person ?a - aircraft),board)", "=", 0),
  m.arithm("sup((in person2 plane2),(board person2 plane2 city1))", "=", 0));
```

Figure 8: A simple example in Choco for the constraints of Figure 6.

are correctly learned (although the temporal annotation of some of them is incorrect), but some effects are unnecessarily learned. We will discuss this in the next Section.

## 5. Experimental Evaluation

We evaluate the quality of our approach from a syntactic+semantic perspective, as typically done in literature. On the one hand, we assess the learning of the PDDL2.1 temporal action model in itself. On the other hand, we restrict the action model to compare our effectiveness *vs.* ARMS and FAMA, two benchmarks for learning classical action models.

We have selected 14 IPC domains[5], which have a *simple-time* version with features we can parse, and solved random problems by using LPG [44]. Using a wide selection of domains helps us to obtain more general results for different application settings. It is important to note that we do not require optimal plans, as the quality of the plans has no impact in the learning process like in other approaches (e.g. LOCM and NLOCM). The use of a CS approach allows us to learn common structures based on the constraints imposed by the observed plans, no matter their quality.

We run a two-fold cross-validation evaluation, where the input collection of plans is partitioned into two sets: up to 10 plans for training and 20 plans for testing. We create three learning scenarios $\mathcal{L}_1$, $\mathcal{L}_5$ and $\mathcal{L}_{10}$ with a collection of 1, 5 and 10 different plans, respectively. We repeat each scenario 20 times, thus creating 20 learning tasks with 1 plan ($\mathcal{L}_1$), 20 tasks with 5 plans ($\mathcal{L}_5$) and so on. Note that each learning task uses a different collection of plans, e.g. each $\mathcal{L}_1$-task uses one different plan as input and it is tested on a different collection of plans. For simplicity, static information is removed from the learning tasks and mutex information is incomplete and only given when it is intuitive from the domain, typically when two objects cannot be at different states simultaneously. The

[5]More information in `https://www.icaps-conference.org/competitions`.

```
(:durative-action board
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration 20)                    ;correct
 :condition (and (at start (at ?p ?c))         ;correct
                 (at start (at ?a ?c)))        ;originally it's over all
 :effect (and (at end (not (at ?p ?c)))        ;originally it's at start
              (at end (in ?p ?a))              ;correct
              (at end (increase (cost) 1))))   ;incorrect
(:durative-action refuel
 :parameters (?a - aircraft ?c - city ?l1 ?l2 - flevel)
 :duration (= ?duration 70)                    ;incorrect
 :condition (and (at start (at ?a ?c))         ;originally it's over all
                 (at start (fuel-level ?a ?l1)) ;correct
                 (at start (next ?l1 ?l2)))    ;correct
 :effect (and (at end (next ?l1 ?l2))          ;unnecessary
              (at end (fuel-level ?a ?l2))     ;correct
              (at end (not (fuel-level ?a ?l1))) ;correct
              (at end (increase (cost) 5))))   ;correct
(:durative-action zoom
 :parameters (?a - aircraft ?c1 ?c2 - city ?l1 ?l2 ?l3 - flevel)
 :duration (= ?duration 95)                    ;incorrect
 :condition (and (at start (at ?a ?c1))        ;correct
                 (at start (next ?l2 ?l1))     ;correct
                 (at start (fuel-level ?a ?l1)) ;correct
                 (at start (next ?l3 ?l2)))    ;correct
 :effect (and (at end (at ?a ?c2))             ;correct
              (at end (fuel-level ?a ?l3))     ;correct
              (at end (not (at ?a ?c1)))       ;originally it's at start
              (at end (not (fuel-level ?a ?l1))) ;correct
              (at end (not (fuel-level ?a ?l2))) ;unnecessary
              (at end (increase (cost) 10))))  ;correct
```

Figure 9: A model learned by Choco for the three operators of Figure 1 of the *zenotravel* domain.

| | $\|\mathcal{O}\|$ | $\|c+e\|$ | $\|\alpha(\mathcal{O})\|$ | $\|\pi\|$ | $\|\mathcal{I}\|$ | $\|\mathcal{G}\|$ | $\mathcal{L}_1$ $\|\mathcal{X}\|$ $\|\mathcal{C}\|$ | $\mathcal{L}_5$ $\|\mathcal{X}\|$ $\|\mathcal{C}\|$ | $\mathcal{L}_{10}$ $\|\mathcal{X}\|$ $\|\mathcal{C}\|$ |
|---|---|---|---|---|---|---|---|---|---|
| depots | 5 | 37 | 36 | 8 | 30 | 2 | 380 7693 | 2843 175363 | 4051 166768 |
| driverlog | 6 | 26 | 24 | 8 | 33 | 4 | 270 8462 | 1151 41533 | 2225 83748 |
| elevator | 4 | 13 | 10 | 8 | 20 | 2 | 167 4310 | 769 23207 | 1683 51431 |
| ferry | 3 | 14 | 11 | 11 | 16 | 6 | 274 38789 | 1253 209181 | 2608 466196 |
| floortile | 7 | 37 | 32 | 9 | 39 | 4 | 337 8183 | 1021 20546 | 2001 41728 |
| gripper | 3 | 14 | 10 | 8 | 13 | 3 | 192 9284 | 818 45895 | 1485 77715 |
| hanoi | 1 | 7 | 12 | 4 | 36 | 2 | 329 13783 | 1741 92842 | 3243 161639 |
| npuzzle | 1 | 6 | 4 | 3 | 40 | 12 | 153 1191 | 681 5214 | 1309 9483 |
| openstacks | 3 | 17 | 17 | 16 | 50 | 6 | 600 36542 | 2727 158670 | 4332 234651 |
| pathways | 5 | 19 | 16 | 7 | 65 | 1 | 227 3241 | 1051 17903 | 2135 36478 |
| pegsol | 1 | 9 | 6 | 4 | 118 | 31 | 367 3075 | 1780 15732 | 3615 33532 |
| satellite | 5 | 20 | 18 | 11 | 19 | 4 | 305 25911 | 1371 110831 | 2680 200201 |
| visitall | 1 | 4 | 4 | 14 | 115 | 12 | 465 7466 | 2248 37020 | 4415 71425 |
| zenotravel | 5 | 24 | 18 | 12 | 27 | 5 | 306 9511 | 1410 47243 | 2655 93759 |

Table 2: Size of the PDDL2.1 domains (with no static predicates), plans and average number of variables and constraints for the $\mathcal{L}_1$, $\mathcal{L}_5$ and $\mathcal{L}_{10}$ learning scenarios.

20*3=60 learning tasks are solved in satisfaction mode and we always select the first solution, so we obtain 20 models per learning scenario to extract average results. The solving time was limited to 300s on an Intel i5-6400@2.70GHz with 8GB of RAM.

Table 2 summarizes the size of the domains *w.r.t.* the number of operators ($|\mathcal{O}|$), number of conditions+effects to learn ($|c+e|$) and alphabet size of all the operators ($|\alpha(\mathcal{O})|$), where the number of potential candidates is three times this value. We also show the average number of actions in the input plans ($|\pi|$), the average size of the initial full states ($|\mathcal{I}|$) and partial goal states ($|\mathcal{G}|$); note that $|\mathcal{I}| > |\mathcal{G}|$ because $\mathcal{I}$ represents a full state and $\mathcal{G}$ a partial one. Finally, we show the average number of variables ($|\mathcal{X}|$) and constraints ($|\mathcal{C}|$) per learning scenario to present the size and scalability of our formulation.

### 5.1. Quality Evaluation

#### 5.1.1. Syntactic evaluation (precision and recall)

The rationale for the syntactic evaluation is to calculate how similar is the learned model $\mathcal{O}(\mathcal{L}_n)$ to the reference model $\mathcal{O}$ from a very strict point of view: both $\mathcal{O}(\mathcal{L}_n)$ and $\mathcal{O}$ should be equal, condition by condition and effect by effect. Let *TP, FP, TN, FN* be true positive, false positive, true negative, and false negative, respectively, in $\mathcal{O}(\mathcal{L}_n)$ calculated over the conditions+effects in $\mathcal{O}$. Precision is the fraction of relevant instances among the learned ones: *precision=TP/(TP+FP)*, which gives us an idea on the soundness of $\mathcal{O}(\mathcal{L}_n)$. Recall is the fraction of the total amount of relevant instances that are learned: *recall=TP/(TP+FN)*, which gives us a notion of the completeness of $\mathcal{O}(\mathcal{L}_n)$. In perfect learning, *precision=recall=1*.

Table 3 depicts the average precision and recall scores of the 20 tasks in the three learning scenarios. We show two experiments per domain, organized in two rows. In the first row, we analyze the simplest case where the duration of the operators is given in the input plans, as typically in PDDL2.1 plans (see Figure 2). In the second row, we analyze a more realist case where the durations are unknown and observed with $\pm 10\%$ of noise. Dealing with completely unspecified durations is uncommon, as we always have some hint or intuition on them. However, we have tested higher values of noise up to fully unknown durations and the learned models are almost identical. We do not show those results as the solving time increases in big domains and some learning tasks cannot be solved within the 300s deadline. In other words, higher values of noise do not have a significant impact *w.r.t.* model quality but they can significantly affect the solving times.

Additionally, we show the results of Table 3 in an $x/y$ form, where $x$ is calculated *w.r.t. TP* conditions/effects that appear and are temporally annotated identically in both $\mathcal{O}(\mathcal{L}_n)$ and $\mathcal{O}$, whereas $y$ is calculated in a more relaxed (and classical) way, i.e., *w.r.t. TP* conditions/effects that appear correctly in both $\mathcal{O}(\mathcal{L}_n)$ and $\mathcal{O}$, no matter if their temporal annotation matches exactly; e.g., an

*at start* effect in the reference model only counts as one *TP* in $x$ if it is learned as an *at start* effect, but it counts as one *TP* in $y$ no matter if it is learned as an *at start* or *at end* effect. The $y$ values are very interesting, as they indirectly reflect (and simulate) the learning of classical action models, with only one type of temporal annotation for (pre)conditions and effects. Obviously, $x \leq y$. Finally, we show the ratio of costs that are correctly learned, which is the same no matter whether the duration is known or observed with noise.

We can extract several conclusions from Table 3:

- The average precision for the conditions is over 0.6 and for the effects 0.7. The average recall for the condition is around 0.7 and for the effects 0.75. Although we do not show the variance of the results in the Table, they are also very good: around 0.02 in most domains.

- Using noisy durations entails very little difference in the quality of learning, and this is a general implication of our experiments. This means that our formulation is robust to deal with that type of uncertainty. As commented above, higher values of noise and even fully unknown durations have not shown significant differences in the models learned. It is important to note that learning the exact durations is almost impossible (even for a human expert) because they can always be learned as shorter values than the real ones, which is also an important practical implication.

- The intuition that learning just *which* ($y$ values) is easier than learning also *when* ($x$ values) is true, no matter the learning scenario. This is more noticeable in the conditions than in the effects, because there are three types of conditions but just two types of effects. In the precision and recall, the difference is around 0.2 for the conditions and less than 0.1 for the effects. The reason for this is that a pure syntax-based measure returns misleading results; e.g., one condition can be learned as *at start* when it is *over all* in the reference model, or vice versa.

- Using multiple plans for learning ($\mathcal{L}_5$ and $\mathcal{L}_{10}$) returns better results on

Table 3: Precision, recall and cost ratio scores for the $\mathcal{L}_1$, $\mathcal{L}_5$ and $\mathcal{L}_{10}$ learning scenarios.

| | $\mathcal{L}_1$ | | | | | $\mathcal{L}_5$ | | | | | $\mathcal{L}_{10}$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | | Recall | | Cost | Precision | | Recall | | Cost | Precision | | Recall | | Cost |
| | Conds | Effs | Conds | Effs | ratio | Conds | Effs | Conds | Effs | ratio | Conds | Effs | Conds | Effs | ratio |
| depots | 0.34/0.63 | 0.53/0.71 | 0.41/0.74 | 0.56/0.75 | 0.01 | 0.36/0.59 | 0.45/0.55 | 0.39/0.64 | 0.47/0.58 | 0.02 | 0.36/0.57 | 0.46/0.56 | 0.40/0.64 | 0.43/0.52 | 0.20 |
| | 0.31/0.59 | 0.48/0.71 | 0.39/0.74 | 0.51/0.75 | | 0.37/0.58 | 0.38/0.54 | 0.41/0.65 | 0.40/0.58 | | 0.37/0.58 | 0.38/0.54 | 0.41/0.65 | 0.40/0.57 | |
| driverlog | 0.47/0.71 | 0.70/0.70 | 0.60/0.91 | 0.81/0.81 | 0.00 | 0.42/0.66 | 0.64/0.64 | 0.55/0.87 | 0.80/0.80 | 0.52 | 0.40/0.64 | 0.65/0.65 | 0.53/0.85 | 0.79/0.79 | 0.62 |
| | 0.45/0.68 | 0.66/0.66 | 0.59/0.89 | 0.78/0.78 | | 0.46/0.69 | 0.64/0.64 | 0.60/0.90 | 0.81/0.81 | | 0.46/0.69 | 0.64/0.64 | 0.60/0.90 | 0.82/0.82 | |
| elevator | 0.59/1 | 0.87/0.87 | 0.50/0.84 | 1/1 | 0.01 | 0.56/0.96 | 0.83/0.83 | 0.47/0.80 | 0.97/0.97 | 0.40 | 0.60/1 | 0.88/0.88 | 0.50/0.83 | 1/1 | 0.49 |
| | 0.59/1 | 0.87/0.87 | 0.50/0.84 | 1/1 | | 0.56/0.96 | 0.83/0.83 | 0.47/0.80 | 0.97/0.97 | | 0.60/1 | 0.88/0.88 | 0.50/0.83 | 1/1 | |
| ferry | 0.50/0.75 | 1/1 | 0.67/1 | 0.75/0.75 | 0.10 | 0.50/0.75 | 1/1 | 0.67/1 | 0.75/0.75 | 0.33 | 0.50/0.75 | 1/1 | 0.67/1 | 0.75/0.75 | 0.33 |
| | 0.50/0.75 | 1/1 | 0.67/1 | 0.75/0.75 | | 0.50/0.75 | 1/1 | 0.67/1 | 0.75/0.75 | | 0.50/0.75 | 1/1 | 0.67/1 | 0.75/0.75 | |
| floortile | 0.41/0.68 | 0.22/0.64 | 0.48/0.78 | 0.23/0.75 | 0.00 | 0.74/0.87 | 0.57/0.82 | 0.77/0.91 | 0.62/0.89 | 0.10 | 0.73/0.94 | 0.63/0.96 | 0.75/0.96 | 0.65/0.99 | 0.81 |
| | 0.45/0.68 | 0.25/0.63 | 0.50/0.77 | 0.27/0.68 | | 0.66/0.80 | 0.60/0.87 | 0.71/0.87 | 0.57/0.83 | | 0.83/0.97 | 0.72/1 | 0.85/1 | 0.72/1 | |
| gripper | 0.52/0.85 | 0.68/0.68 | 0.52/0.85 | 0.49/0.49 | 0.02 | 0.65/0.98 | 0.97/0.97 | 0.65/0.98 | 0.84/0.84 | 0.33 | 0.67/1 | 1/1 | 0.67/1 | 0.88/0.88 | 0.33 |
| | 0.52/0.85 | 0.68/0.68 | 0.52/0.85 | 0.49/0.49 | | 0.64/0.98 | 0.96/0.96 | 0.82/0.98 | 0.82/0.82 | | 0.67/1 | 1/1 | 0.67/1 | 0.88/0.88 | |
| hanoi | 0.91/1 | 0.33/0.46 | 0.83/0.92 | 0.46/0.65 | 1 | 0.93/1 | 0.24/0.35 | 0.67/0.72 | 0.51/0.75 | 1 | 1/1 | 0.22/0.33 | 0.67/0.67 | 0.5/0.75 | 1 |
| | 0.91/1 | 0.33/0.46 | 0.83/0.92 | 0.46/0.65 | | 0.93/1 | 0.24/0.35 | 0.67/0.72 | 0.51/0.75 | | 1/1 | 0.22/0.33 | 0.67/0.67 | 0.5/0.75 | |
| npuzzle | 1/1 | 1/1 | 1/1 | 0.96/0.96 | 1 | 1/1 | 1/1 | 1/1 | 1/1 | 1 | 1/1 | 1/1 | 1/1 | 1/1 | 1 |
| | 1/1 | 1/1 | 1/1 | 0.96/0.96 | | 1/1 | 1/1 | 1/1 | 1/1 | | 1/1 | 1/1 | 1/1 | 1/1 | |
| openstacks | 0.76/0.76 | 0.71/0.71 | 0.76/0.76 | 0.92/0.92 | 0.00 | 0.70/0.70 | 0.87/0.87 | 0.70/0.70 | 0.79/0.79 | 0.33 | 0.57/0.57 | 0.68/0.68 | 0.56/0.56 | 0.69/0.69 | 0.33 |
| | 0.57/0.69 | 0.59/0.70 | 0.68/0.82 | 0.81/0.95 | | 0.43/0.57 | 0.89/0.89 | 0.43/0.57 | 0.80/0.80 | | 0.39/0.43 | 0.74/0.74 | 0.43/0.47 | 0.87/0.87 | |
| pathways | 0.60/0.63 | 0.46/0.55 | 0.77/0.81 | 0.42/0.51 | 0.00 | 0.76/0.76 | 0.64/0.64 | 0.84/0.84 | 0.57/0.57 | 0.39 | 0.84/0.84 | 0.74/0.74 | 0.85/0.85 | 0.64/0.64 | 0.78 |
| | 0.63/0.63 | 0.50/0.50 | 0.81/0.81 | 0.46/0.46 | | 0.76/0.76 | 0.64/0.64 | 0.84/0.84 | 0.57/0.57 | | 0.81/0.83 | 0.70/0.74 | 0.84/0.86 | 0.61/0.64 | |
| pegsol | 1/1 | 1/1 | 1/1 | 1/1 | 1 | 1/1 | 1/1 | 1/1 | 1/1 | 1 | 1/1 | 1/1 | 1/1 | 1/1 | 1 |
| | 1/1 | 1/1 | 1/1 | 1/1 | | 1/1 | 1/1 | 1/1 | 1/1 | | 1/1 | 1/1 | 1/1 | 1/1 | |
| satellite | 0.27/0.69 | 0.43/0.57 | 0.30/0.90 | 0.43/0.57 | 0.00 | 0.27/0.69 | 0.56/0.65 | 0.30/0.90 | 0.56/0.70 | 0.42 | 0.28/0.67 | 0.56/0.69 | 0.32/0.90 | 0.56/0.69 | 0.47 |
| | 0.27/0.69 | 0.43/0.57 | 0.30/0.90 | 0.43/0.57 | | 0.27/0.69 | 0.56/0.70 | 0.30/0.90 | 0.56/0.70 | | 0.27/0.69 | 0.57/0.71 | 0.30/0.90 | 0.57/0.71 | |
| visitall | 0.50/0.50 | 1/1 | 1/1 | 1/1 | 1 | 0.50/0.50 | 1/1 | 1/1 | 1/1 | 1 | 0.50/0.50 | 1/1 | 1/1 | 1/1 | 1 |
| | 0.50/0.50 | 1/1 | 1/1 | 1/1 | | 0.50/0.50 | 1/1 | 1/1 | 1/1 | | 0.50/0.50 | 1/1 | 1/1 | 1/1 | |
| zenotravel | 0.66/0.99 | 0.79/0.96 | 0.64/0.95 | 0.78/0.94 | 0.01 | 0.70/1 | 0.76/0.95 | 0.70/1 | 0.77/0.96 | 0.58 | 0.70/1 | 0.74/0.94 | 0.70/1 | 0.78/0.99 | 0.60 |
| | 0.63/0.93 | 0.67/0.84 | 0.62/0.92 | 0.69/0.87 | | 0.69/0.99 | 0.75/0.93 | 0.68/0.98 | 0.75/0.94 | | 0.70/1 | 0.74/0.94 | 0.70/1 | 0.78/0.99 | |
| Average | 0.61/0.80 | 0.69/0.78 | 0.68/0.89 | 0.70/0.79 | 0.30 | 0.65/0.82 | 0.75/0.81 | 0.69/0.88 | 0.76/0.83 | 0.53 | 0.65/0.82 | 0.75/0.82 | 0.69/0.88 | 0.76/0.84 | 0.64 |
| | 0.60/0.79 | 0.68/0.76 | 0.67/0.89 | 0.69/0.78 | | 0.63/0.81 | 0.75/0.81 | 0.69/0.87 | 0.75/0.82 | | 0.65/0.82 | 0.76/0.82 | 0.69/0.88 | 0.78/0.86 | |

average than $\mathcal{L}_1$, which represents an interesting improvement *w.r.t.* the work in [26]. However, there is not a significant improvement in $\mathcal{L}_{10}$ *vs.* $\mathcal{L}_5$, which means that our approach does not require large datasets of input plans. A practical implication here is that we need a sufficient number of plans for learning a model, but when that number is exceeded no further improvements in the learned model are achieved. Actually, using many input plans might introduce slight dispersion in some domains like *driverlog*, *hanoi* and *openstacks*, particularly in the recall. The reason for this is that the resulting CSP has several solutions and in these domains the solver returns a solution more rapidly (the problem is more constrained in $\mathcal{L}_5$ and $\mathcal{L}_{10}$ than in $\mathcal{L}_1$) that does not always represent the most complete model. Consequently, in some specific domains, the precision and recall can show worse results when dealing with more input plans. This is an open issue yet, as we have not found a proper way (or metric) that successfully solves this drawback in these particular domains. On the other hand, using large datasets of plans shows beneficial for learning the costs: clearly, the cost ratio always improves when using more plans. Nevertheless, it is important to note that there are domains where the costs cannot be learned precisely. If some operators always appear in pairs (e.g., `board` and `debark` in *zenotravel*, as no person remains in an aircraft forever), this leads to an inconclusive C12 constraint: we can learn the cost of the two operators as a pair, but not individually. The practical consequence here is that this remains unsolvable no matter the size of the dataset.

- Learning the perfect model is infrequent but still possible for small domains (*npuzzle*, *pegsol* and *visitall*), which shows the power of a CS approach. In the domains where the perfect model is not learned, we could try to merge several models. For instance, is it possible, and sensible, to merge (part of) the 20 solutions of every learning scenario in just one solution? The unification of models is appealing but it entails many difficulties when the models show contradictions, e.g. the same condition

is learned as *at start* in one model and *overall* in another, or the same effect is learned as *at start* and *at end* in different models. Using the most restrictive model seems a reasonable idea but it may not lead to a better model and requires deeper investigation.

### *5.1.2. Semantic evaluation*

There is not a unique reference model in temporal planning, so strict similarity measures can be unfair. Actually, the learned model might be a reformulation of the reference model if some conditions/effects are interchangeable or unnecessarily learned but still correct. In other words, both models are syntactically different but can be semantically equivalent. By equivalent we mean that the learned model is valid for an unseen plan because it reproduces, with no contradictions or inconsistencies, the plan observations. Therefore, the rationale for the semantic evaluation is to calculate if the model $\mathcal{O}(\mathcal{L}_n)$ is still valid for the plans calculated by the reference model $\mathcal{O}$; i.e., perhaps the two models look different but they capture the same physics of the problem.

Formally, let us assume a new problem $\rho$ and its corresponding valid plan $\pi(\rho)$ created from a reference model $\mathcal{O}$. If we now instantiate actions in $\pi(\rho)$ according to the operators in $\mathcal{O}(\mathcal{L}_n)$ and all the constraints imposed by $\pi(\rho)$ are satisfied, then $\mathcal{O}(\mathcal{L}_n)$ is semantically equivalent to $\mathcal{O}$, and we count this as a hit in *TP*. Thus, we define the fraction of the instances in a test dataset that are equivalent as $ratio=TP/|dataset|$. If *ratio=1* we have learned a model that satisfies all plans in the dataset.

We have defined 20 learning tasks per learning scenario, which means learning 20 models. Each model is semantically evaluated *vs.* a dataset of 20 new testing plans, i.e., 20*20=400 evaluations per scenario. Table 4 depicts the average ratios, which are shown as $x/y$ values: now $x$ is calculated when the duration of operators is fully known, and $y$ when durations are observed with $\pm 10\%$ of noise. The ratios are very good, specially in $\mathcal{L}_5$ and $\mathcal{L}_{10}$, where the perfect ratio is found in over half of the domains. This shows again the advantages of using multiple plans for learning *vs.* using only one plan like in [26].

| | $\mathcal{L}_1$ | $\mathcal{L}_5$ | $\mathcal{L}_{10}$ |
|---|---|---|---|
| depots | 0.55/0.42 | 0.87/0.92 | 0.77/0.86 |
| driverlog | 0.65/0.72 | 1/0.97 | 1/1 |
| elevator | 1/1 | 1/1 | 1/1 |
| ferry | 1/1 | 1/1 | 0.96/0.96 |
| floortile | 0.17/0.25 | 0.24/0.14 | 0.42/0.20 |
| gripper | 0.83/0.83 | 1/1 | 1/1 |
| hanoi | 0.27/0.27 | 0.78/0.78 | 0.90/0.90 |
| npuzzle | 0.90/0.90 | 1/1 | 1/1 |
| openstacks | 1/0.70 | 1/1 | 1/0.98 |
| pathways | 0.63/0.63 | 0.88/0.88 | 0.99/0.99 |
| pegsol | 1/1 | 1/1 | 1/1 |
| satellite | 1/1 | 1/1 | 1/1 |
| visitall | 1/1 | 1/1 | 1/1 |
| zenotravel | 0.91/0.64 | 0.81/0.79 | 0.96/0.96 |
| Average | 0.78/0.74 | 0.90/0.89 | 0.93/0.91 |

Table 4: Average ratio for the semantic evaluation in $\mathcal{L}_1$, $\mathcal{L}_5$ and $\mathcal{L}_{10}$ of the learned model *vs.* the test dataset.

Although not show in the Table, the variance of the results is also remarkable: less than 0.1 in most domains and never more than 0.2. Again, there is not a significant difference when using noisy or even unknown durations. The main conclusion is that the quality of our learned models is very good from a semantic perspective. The practical implication for this is that the physics of the problem is, on average, successfully learned.

### *5.2. Comparison to ARMS and FAMA*

The rationale for this section is that, to our knowledge, there is no previous approach in planning capable of learning temporal action models to compare with. However, learning classical action models, without durations or costs, has been very successful in ARMS and FAMA. This makes reasonable to analyze the results of a complete comparison.

In this section, we compare our CS approach to ARMS and FAMA to help its positioning *w.r.t.* these two successful approaches. We are specially interested in assessing whether the precision and recall values are comparable to those achieved by two benchmarks for learning in classical planning. To simulate classical models, all durations (X1) and costs (X2) are equal and we restrict the domain of conditions (X3) and effects (X4) to *at_start* and *at_end*, respectively. This shows how our formulation subsumes (and generalizes) the learning of classical models. ARMS and FAMA show important gain when observing intermediate states. Thus, they define an observability degree that measures the probability of observing a predicate $p$ in a state at time $t$. We include this in our formulation by using dummy $\mathsf{obs}(p, t)$.

For our comparison, we use the results of ARMS and FAMA as provided in [22] and for the eight domains with a *simple-time* version that we can parse, i.e., *driverlog*, *ferry*, *floortile*, *gripper*, *npuzzle*, *satellite*, *visitall* and *zenotravel*. Here, we learn from 10 sequential plans ($\mathcal{L}_{10}$), with 10 actions each, per domain, where static information is not removed from the domain to make the comparison completely fair. Figures 10 and 11 depict the average results for precision and recall, respectively. For our CS approach, we show the results for both including (and not) mutex information, i.e., CS-mutex and CS-no mutex, respectively. Mutex reasoning is only relevant for learning negative effects in absence of intermediate observations, as shown in the recall with 0% of observability. In a scenario of null observability such reasoning shows very powerful. Apart from this, there is not a significant difference between CS-mutex and CS-no mutex for precision and recall as negative effects can be learned from the observations. Actually, the results are equal after 60% of observability.

Our results are very competitive: FAMA is slightly better for precision but not for recall, and our precision/recall is notably higher than in ARMS. Additionally, we support more features such as temporal action models, noisy durations and cost learning. The main practical implication is that our formulation for learning temporal models is very general and flexible. It can be easily adapted for classical models, where it provides very steady results even for minimal ob-

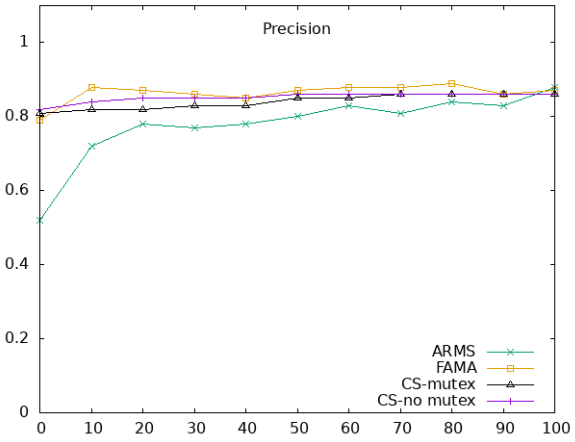


Figure 10: Average precision for 0-100% observability.

servability degrees, and remain highly competitive.

## 6. Conclusions

Learning action models from multiple plans has proved successful for classical planning, but there is an important void for supporting the temporal annotations of durative actions and their costs. Since learning temporal models seems the natural evolution of learning classical models, this work can be interpreted as a first step to build both the academic and engineering foundations for learning in temporal planning settings. From an academic point of view, it pushes forward the research on learning more expressive action models while bridging the gap with other approaches for constraint acquisition. From an engineering point of view, it provides a flexible CS formulation that can be easily adapted to identify complex structures in industrial applications.

We have proposed a solver-independent CS formulation, which automatically compiles a learning task, with several advantages:

- It is designed for PDDL2.1 but accommodates other levels of expressiveness, higher (e.g., Allen's relations) or lower (e.g., classical models) ones

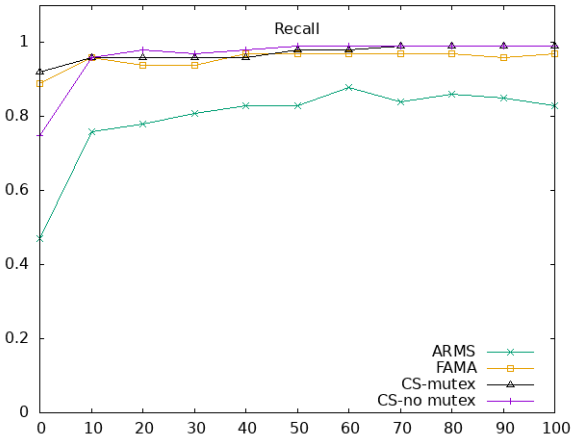


Figure 11: Average recall for 0-100% observability.

by modifying the constraints. This is an interesting result as it fosters a general approach for learning in different planning contexts.

- We can include additional input knowledge such as mutex information, which reduces the necessity of observations and helps to learn more complete (recall) models. The main interpretation of this is that, if intermediate state observability is impossible, mutex reasoning is essential for learning negative effects.

- The formulation is general enough to learn from a flexible number of plans, no matter their quality: from a single plan ($\mathcal{L}_1$, *aka* one-shot learning) to a collection of $n$ plans ($\mathcal{L}_n$), all of which are completely satisfied. This generality is a clear advantage and shows very useful in real-world applications, where the plans can be limited and/or expensive to obtain, particularly if they require repetitive processes. From a solving time perspective, our formulation includes a little overload in comparison to [26] in $\mathcal{L}_1$ because of the operator-action duality representation, but this pays off when more input plans are available, which leads to a practical improvement in the precision/recall. Our experiments and findings reveal that we do not need many input plans nor they need to be particularly long. Actually, dealing

with many plans in all domains is not always a good idea, as they lead to a higher dispersion when learning in some specific domains. As a practical result, 5-10 input plans with no more than 10 actions each are enough to learn good quality models and costs, though learning the exact duration, as given in the reference model, is unlikely. After all, the duration can always be shorter and have more time for no-operations.

- Our formulation can be extended with a metric to express user's preferences, thus moving to a constraint optimization problem. Although this is plausible, our investigation reveals that such a metric does not have a significant impact in the quality of learning. This is an important research finding, because the intention of the domain expert cannot be always defined as a metric. A possible option that we have studied is to find several models per learning task and return the most repeated one, or try to unify several models into the most restrictive one. This opens a future line of research for unification and model merging that requires a more profound analysis.

- Although we learn from empty operators and complete plans, the formulation is also appropriate to deal with partial action models and incomplete fragments of plans. In other words, restricting some variables leads us to a unified formulation that bridges the gap between learning and validation. This still requires further investigation.

- Our CS formulation can be solved by Satisfiability Modulo Theories, which is part of our future work.

For the future, we also want to learn meta-models in an incremental way, by iteratively learning from an increasing number of input plans. From an industrial perspective, this can indirectly lead to the reformulation of existing models to validate, simplify and unify them. On the one hand, the human domain definition is not always coherent *w.r.t.* the temporal annotation (e.g. an *at start* condition that should be *over all*, as it happens in *driverlog* and *zenotravel* IPC

domains). Analogously, some *at start* effects should be *at end*, and vice versa, as it happens in *satellite* and *zenotravel* domains. On the other hand, some (down-level) actions can be generalized into (top-level) macro-actions, which makes the action model more compact and efficient; this is specially interesting in Hierarchical Task Network (HTN) planning.


## Acknowledgments

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


## References


[1] A. Appiah, X. Zhang, B. Ayawli, F. Kyeremeh, Long short-term memory networks based automatic feature extraction for photovoltaic array fault diagnosis, IEEE Access 7 (2019) 30089–30101. doi:https://doi.org/10.1109/ACCESS.2019.2902949.

[2] S. Jialin Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on Knowledge and Data Engineering 22(10) (2010) 1345–1359. doi:https://doi.org/10.1109/TKDE.2009.191.

[3] G. Kou, J. Yue, Y. Ma, Q. Wang, Z. Zhang, SAR image invariant feature extraction by anisotropic diffusion and multi-gray level simplified PCNN, IEEE Access 7 (2019) 47135–47142. doi:https://doi.org/10.1109/ACCESS.2019.2906362.

[4] C. Lauretti, F. Cordella, A. L. Ciancio, E. Trigili, J. M. Catalan, F. J. Badesa, S. Crea, S. M. Pagliara, S. Sterzi, N. Vitiello, N. Garcia Aracil, L. Zollo, Learning by demonstration for motion planning of upper-limb exoskeletons, Frontiers in Neurorobotics 12 (2018) 1–5. doi:https://doi.org/10.3389/fnbot.2018.00005.

[5] L. Liu, S. Wang, B. Hu, Q. Qiong, D. Rosenblum, Learning structures of interval-based bayesian networks in probabilistic generative model for human complex activity recognition, Pattern Recognition 81 (2018) 545–561. doi:https://doi.org/10.1016/j.patcog.2018.04.022.

[6] Y. Xu, J. Tsujii, E. I.-C. Chang, Named entity recognition of follow-up and time information in 20000 radiology reports, Journal of the American Medical Informatics Association 19 (5) (2012) 792–799. doi:https://doi.org/10.1136/amiajnl-2012-000812.

[7] S. Sohrabi, AI planning for enterprise: Putting theory into practice, in: Proc. International Joint Conference on AI (IJCAI-2019), 2019, pp. 6408–6410. doi:https://doi.org/10.24963/ijcai.2019/897.

[8] M. Vukovic, S. Gerard, R. Hull, M. Katz, L. Shwartz, S. Sohrabi, C. Muise, J. Rofrano, A. Kalia, J. Hwang, Y. D., J. Jie, M. Zhuoxuan, Towards automated planning for enterprise services: Opportunities and challenges, in: Proc. International Conference on Service-Oriented Computing (ICSOC-2019), 2019, pp. 64–68. doi:https://doi.org/10.1007/978-3-030-33702-5_6.

[9] R. Fikes, N. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (1971) 189–208. doi:https://doi.org/10.1016/0004-3702(71)90010-5.

[10] H. Geffner, Functional STRIPS: a more flexible language for planning and problem solving, Logic-Based Artificial Intelligence (2000). doi:https://doi.org/10.1007/978-1-4615-1567-8_9.

[11] E. Pednault, ADL: Exploring the middle ground between strips and the situation calculus, in: Proc. Int. Conference on Principles of Knowledge Representation and Reasoning (KR-89), Morgan Kaufmann, San Francisco, CA, 1989, pp. 324–332. doi:https://dl.acm.org/doi/10.5555/112922.112954.

[12] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL - The Planning Domain Definition Language, AIPS-98 Planning Competition Committee (1998). doi:http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212.

[13] M. Fox, D. Long, PDDL2.1: an extension to PDDL for expressing temporal planning domains, Journal of Artificial Intelligence Research 20 (2003) 61–124. doi:https://doi.org/10.1613/jair.1129.

[14] S. Kambhampati, Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models, in: Proceedings of the National Conference on Artificial Intelligence (AAAI-07), Vol. 22(2), 2007, pp. 1601–1604.

[15] H. H. Zhuo, Q. Yang, D. H. Hu, L. Li, Learning complex action models with quantifiers and logical implications, Artificial Intelligence 174 (18) (2010) 1540–1569. doi:https://doi.org/10.1016/j.artint.2010.09.007.

[16] V. Strobel, A. Kirsch, Planning in the wild: Modeling tools for pddl, in: Proc. Joint German/Austrian Conference on Artificial Intelligence (KI-2014), 2014, pp. 273–284. doi:https://doi.org/10.1007/978-3-319-11206-0_27.

[17] S. Jiménez, T. De la Rosa, S. Fernández, F. Fernández, D. Borrajo, A review of machine learning for automated planning, The Knowledge Engineering Review 27 (4) (2012) 433–467. doi:https://doi.org/10.1017/S026988891200001X.

[18] M. Cashmore, A. Collins, B. Krarup, S. Krivic, D. Magazzeni, D. Smith, Towards explainable ai planning as a service, in: Proc. 2nd ICAPS Workshop on Explainable Planning (XAIP-2019), 2019, pp. 1–9. doi:https://arxiv.org/abs/1908.05059.

[19] M. Ramirez, H. Geffner, Plan recognition as planning, in: Proc. International Joint Conference on AI (IJCAI-2009), 2016, pp. 1778–1783.

[20] R. Pereira, N. Oren, F. Meneguzzi, Landmark-based approaches for goal recognition as planning, Artificial Intelligence 279 (2020) 1–49. doi:https://arxiv.org/abs/1904.11739.

[21] S. Sohrabi, A. Riabov, O. Udrea, Plan recognition as planning revisited, in: Proc. International Joint Conference on AI (IJCAI-2016), 2016, pp. 3258–3264. doi:https://dl.acm.org/doi/10.5555/3061053.3061077.

[22] D. Aineto, S. Jiménez, E. Onaindia, Learning action models with minimal observability, Artificial Intelligence 275 (2019) 104–137. doi:https://doi.org/10.1016/j.artint.2019.05.003.

[23] E. Amir, A. Chang, Learning partially observable deterministic action models, Journal of Artificial Intelligence Research 33 (2008) 349–402. doi:https://doi.org/10.1613/jair.2575.

[24] Q. Yang, K. Wu, Y. Jiang, Learning action models from plan examples using weighted MAX-SAT, Artificial Intelligence 171 (2-3) (2007) 107–143. doi:https://doi.org/10.1016/j.artint.2006.11.005.

[25] W. Cushing, S. Kambhampati, Mausam, D. Weld, When is temporal planning really temporal?, in: Proc. Int. Joint Conference on AI (IJCAI-2007), Morgan Kaufmann Publishers Inc., 2007, pp. 1852–1859. doi:https://dl.acm.org/doi/10.5555/1625275.1625575.

[26] A. Garrido, S. Jimenez, Learning temporal action models via constraint programming, in: Proc. of the 24th European Conference on Artificial Intelligence (ECAI 2020), 2020, pp. 2362–2369. doi:10.3233/FAIA200366.

[27] C. Bessiere, R. Coletta, F. Koriche, B. O'Sullivan, Acquiring constraint networks using a SAT-based version space algorithm, in: Proc. of the AAAI Conference on Artificial Intelligence, 2006, pp. 1565–1568.

[28] N. Beldiceanu, H. Simonis, A constraint seeker: Finding and ranking global constraints from examples, in: Proc. of the Int. Conference on Principles

and Practice of Constraint Programming (CP-2011), 2011, pp. 12–26. doi:https://doi.org/10.1007/978-3-642-23786-7_4.

[29] N. Beldiceanu, H. Simonis, A model seeker: extracting global constraint models from positive examples, in: Proc. of the Int. Conference on Principles and Practice of Constraint Programming (CP-2012), 2012, pp. 141–157. doi:https://doi.org/10.1007/978-3-642-33558-7_13.

[30] C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C. Quimper, T. Walsh, Constraint acquisition via partial queries, in: Proc. International Joint Conference on AI (IJCAI-2013), 2013, pp. 475?–481. doi:https://dl.acm.org/doi/10.5555/2540128.2540198.

[31] S. Benson, Learning action models for reactive autonomous agents, Ph.D. thesis, Stanford University (1996). doi:https://dl.acm.org/doi/book/10.5555/892612.

[32] S. Muggleton, L. Raedt, Inductive logic programming: Theory and methods, The Journal of Logic Programming 19-20 (1994) 629–679. doi:https://doi.org/10.1016/0743-1066(94)90035-3.

[33] G. Sablon, D. Boulanger, Using the event calculus to integrate planning and learning in an intelligent autonomous agent, in: Proc. of the Workshop on Planning Current Trends in AI Planning, 1994, pp. 254–265.

[34] H. H. Zhuo, S. Kambhampati, Action-model acquisition from noisy plan traces, in: International Joint Conference on Artificial Intelligence, IJCAI-13, 2013, pp. 2444–2450. doi:https://dl.acm.org/doi/10.5555/2540128.2540479.

[35] H. H. Zhuo, T. A. Nguyen, S. Kambhampati, Refining incomplete planning domain models through plan traces, in: International Joint Conference on Artificial Intelligence, IJCAI-13, 2013, pp. 2451–2458. doi:https://dl.acm.org/doi/abs/10.5555/2540128.2540480.

[36] S. Cresswell, P. Gregory, Generalised domain model acquisition from action traces, in: Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2011), 2011, pp. 42–49. doi:https://dl.acm.org/doi/abs/10.5555/3038485.3038492.

[37] S. N. Cresswell, T. McCluskey, M. West, Acquiring planning domain models using LOCM, The Knowledge Engineering Review 28(2) (2013) 195–213. doi:https://doi.org/10.1017/S0269888912000422.

[38] P. Gregory, A. Lindsay, Domain model acquisition in domains with action costs, in: Proc. Int. Conference on Automated Planning and Scheduling (ICAPS-2016), 2011, pp. 149–157. doi:https://dl.acm.org/doi/abs/10.5555/3038594.3038613.

[39] J. Kucera, R. Barták, LOUGA: learning planning operators using genetic algorithms, in: Pacific Rim Knowledge Acquisition Workshop, PKAW-18, 2018, pp. 124–138. doi:https://doi.org/10.1007/978-3-319-97289-3_10.

[40] A. Garrido, M. Fox, D. Long, A temporal planning system for durative actions of PDDL2.1, in: F. V. Harmelen (Ed.), Proc. European Conference on AI (ECAI-2002), IOS Press, Amsterdam, 2002, pp. 586–590.

[41] D. McAllester, D. Rosenblitt, Systematic nonlinear planning, in: Proc. 9th Nat. Conference on AI, Anaheim, CA, 1991, pp. 634–639. doi:https://dl.acm.org/doi/10.5555/1865756.1865775.

[42] J. Penberthy, D. Weld, UCPOP: a sound, complete, partial-order planner for ADL, in: Proc. Int. Conference on Principles of Knowledge Representation and Reasoning, Kaufmann, Los Altos, CA, 1992, pp. 103–114. doi:https://dl.acm.org/doi/abs/10.5555/3087223.3087234.

[43] T. Walsh, SAT v CSP, in: Proc. of the Int. Conference on Principles and Practice of Constraint Programming (CP-2000), 2000, pp. 441–456. doi:https://link.springer.com/chapter/10.1007/3-540-45349-0_32.

[44] A. Gerevini, A. Saetti, I. Serina, Planning through stochastic local search and temporal action graphs in LPG, Journal of Artificial Intelligence Research 20 (2003) 239–290. doi:https://doi.org/10.1613/jair.1183.