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

No Author Given

No Institute Given

**Abstract.** This work proposes a novel constraint programming approach for learning durative actions in an expressive temporal planning model with overlapping actions, which makes it suitable for learning in multi-agent environments. We analyze the extreme scenario, where just a single (one-shot) partial observation of the execution of a temporal plan is available, to learn the distribution of conditions/effects and estimate the durations, resulting in a consistent constraint model. Our approach automatically builds a purely declarative formulation that models time-stamps for actions, causal link relationships (conditions and effects), threats and effect interferences that appear in planning. It also accommodates a different range of expressiveness, subsuming the PDDL2.1 temporal semantics. Our formulation is simple but effective, and is not only valid for learning, but also for plan validation, as shown in its evaluation that returns high success ratios. Finally, our formulation is solver-independent, meaning that an arbitrary CSP solver can be used for its resolution.

**Keywords:** One-shot learning action models · Temporal planning · Partial observability · Constraint programming.

## 1 Introduction

*Automated planning* is the model-based approach for the task of selecting the actions that achieve a given set of goals starting from a given initial state. *Classical planning* is the vanilla model for planning and it assumes: fully observable states under a deterministic world, instantaneous actions, and goals that are exclusively referred to the last state reached by a plan [8, 10]. Beyond classical planning, there is a bunch of more expressive planning models that relax the previous assumptions to compute more detailed solutions than classical plans [10].

*Temporal planning* is one of these more expressive planning models, as it relaxes the assumption of instantaneous actions [6]. Temporal actions have durations and conditions/effects that must hold/happen at different times, which means that temporal actions can be executed in parallel and overlap in several ways [4]. Consequently, valid solution plans for temporal planning problems need to indicate the precise time-stamp when an action starts and ends [11].

Despite the potential of state-of-the-art planners, its applicability to the real world is still somewhat limited because of the difficulty of specifying correct and complete planning models [14]. The more expressive the planning model is, the more evident becomes this knowledge acquisition bottleneck, which jeopardizes the usability of AI planning technology. This has led to a growing interest in the planning community for the learning of action models [13]. The objective of this learning task is to compute the actions' conditions and effects that are *consistent* with a set of noiseless observations (defined as some sequence of state changes, input constraints, expert demonstrations or plan traces/logs). Model learning from observation of past behavior provides indirect, but very valuable information to hypothesize the action models, thus helping future planning decisions and recommendations. This is specially interesting for proactive assistants when recognizing activities of multiple (human or software) agents to assist them in their daily activities.

Most approaches for learning planning action models are purely inductive and often require large datasets of observations, e.g. thousands of plan observations to compute a statistically significant model that minimizes some error metric over the observations [15, 17–19]. Defining model learning as an optimization task over a set of observations does not guarantee completeness (the learned model may fail to explain an observation), nor correctness (the states induced by the execution of the plan generated with the model may contain contradictory information). This paper analyzes the application of Constraint Programming for *one-shot learning* of temporal action models, that is, the extreme case of learning action models from a single and partially specified model observed from the execution of a temporal plan. Further, the paper evidences that the learning of action models strongly resembles the task of synthesizing and validating a plan that satisfies all the imposed constraints or, in other words, that is consistent with the noiseless input observations.

While learning an action model for classical planning means computing the actions' conditions and effects that are consistent with the input observations, learning temporal action models extends this to: i) identify how these conditions and effects are temporally distributed in the action execution, and ii) estimate the action duration. As a motivating example, let us assume a logistics scenario. Learning the temporal planning model will allow us: i) to better understand the insights of the logistics in terms of what is possible (or not) and why, because the model is consistent with the observed data; ii) to suggest changes that can improve the model originally created by a human, e.g. re-distributing the actions' conditions, provided they still explain the observations; and iii) to automatically elaborate similar models for similar scenarios, such as public transit for commuters, tourists or people in general in metropolitan areas —*a.k.a.* smart urban mobility.

Learning classical action models has been addressed by different approaches [2]. Since pioneering learning systems like ARMS [18], we have seen systems able to learn action models with quantifiers [1, 22], from noisy actions or states [17, 19], from null state information [3], or from incomplete domain models [20, 21]. But, to our knowledge, none of these systems learns the temporal features. This means that observations may now refer to the execution of overlapping durative actions, which makes our approach suitable for learning in multi-agent environments.

## 2 Background and Terminology

This section formalizes the *classical* and *temporal* planning models that we follow in this work.

### 2.1 Classical Planning

Let $F$ be a set of facts that represent propositional variables. A state $s$ is a full assignment of values to variables, $|s| = |F|$, so the size of the state space is $2^{|F|}$. A *classical planning problem* is a tuple $\langle F, I, G, A\rangle$, where $I$ is the initial state, $G \subseteq F$ is a set of goal conditions over $F$, and $A$ is the set of actions that modify states. We assume that actions are grounded from action schemas or operators, as in PDDL (Planning Domain Definition Language [6, 10]).

Each action $a \in A$ has a set of preconditions $\mathsf{pre}(a)$ and a set of effects $\mathsf{eff}(a)$; $\mathsf{pre}(a), \mathsf{eff}(a) \subseteq F$. $\mathsf{pre}(a)$ must hold before $a$ starts (this is why they are named *pre*conditions), whereas $\mathsf{eff}(a)$ happen when $a$ ends. This way, $a$ is applicable in a state $s$ if $\mathsf{pre}(a) \subseteq s$. When $a$ is executed, a new state, the successor of $s$, is created that results of applying $\mathsf{eff}(a)$ on $s$. Typically, $\mathsf{eff}(a)$ is formed by positive and negative/delete effects. Fig. 1 shows an example of two classical actions for a logistics scenario, from the *driverlog* domain of the International Planning Competition.[1] Action `board-truck` boards a driver on an empty truck at a given location. In `drive-truck` a truck is driven between two locations, provided there is a link between them.

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

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

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

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

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

in the last state. In general terms, classical plans can be sequential plans, where only one action is executed at each planning step, or parallel plans, where several actions can be executed at the same planning step.

### 2.2 Temporal Planning

A *temporal planning problem* is also a tuple $\langle F, I, G, A\rangle$ where $F$, $I$ and $G$ are defined like in classical planning, and $A$ represents the set of *durative actions*. There are several options that allow for a high expressiveness of durative actions. On the one hand, an action can have a fixed duration, a duration that ranges within an interval or a distribution of durations. On the other hand, actions may have conditions/effects at different times, such as conditions that must hold some time before the action starts, effects that happen just when the action starts, in the middle of the action or some time after the action finishes [7].

A popular model for temporal planning is given by PDDL2.1 [6], a language that somewhat restricts temporal expressiveness, which defines a durative action $a$ with the following elements:

- $\mathsf{dur}(a)$, a positive value for the action duration.
- $\mathsf{cond}_s(a), \mathsf{cond}_o(a), \mathsf{cond}_e(a) \subseteq F$. Unlike the *pre*conditions of a classical action, now conditions must hold before $a$ (*at start*), during the entire execution of $a$ (*over all*) or when $a$ finishes (*at end*), respectively. In the simplest case, $\mathsf{cond}_s(a) \cup \mathsf{cond}_o(a) \cup \mathsf{cond}_e(a) = \mathsf{pre}(a)$.[2]
- $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$. Now effects can happen *at start* or *at end* of $a$, respectively, and can still be positive or negative. Again, in the simplest case $\mathsf{eff}_s(a) \cup \mathsf{eff}_e(a) = \mathsf{eff}(a)$.

The semantics of a PDDL2.1 durative action $a$ can be defined in terms of two discrete events, $\mathsf{start}(a)$ and $\mathsf{end}(a) = \mathsf{start}(a) + \mathsf{dur}(a)$. This means that if action $a$ starts on state $s$, $\mathsf{cond}_s(a)$ must hold in $s$; and ending $a$ in state $s'$ means $\mathsf{cond}_e(a)$ holds in $s'$. *Over all* conditions must hold at any state between $s$ and $s'$ or, in other words, throughout interval $[\mathsf{start}(a)..\mathsf{end}(a)]$. Analogously, *at start* and *at end* effects are instantaneously applied at states $s$ and $s'$, respectively —continuous effects are not considered. Fig. 2 shows two durative actions that extend the classical actions of Fig. 1. Now `board-truck` has a fixed duration whereas in `drive-truck` the duration depends on the two locations.

A temporal plan is a set of pairs $\langle (a_1, t_1), (a_2, t_2) \ldots (a_n, t_n)\rangle$. Each $(a_i, t_i)$ pair contains a durative action $a_i$ and $t_i = \mathsf{start}(a_i)$. This temporal plan induces a state sequence formed by the union of all states $\{s_{t_i}, s_{t_i + \mathsf{dur}(a_i)}\}$, where there exists a state $s_0 = I$, and $G \subseteq s_{end}$, being $s_{end}$ the last state induced by the plan. Though a sequential temporal plan is syntactically possible, it is semantically useless. Consequently, temporal plans are always given as parallel plans.

[2] Note that in classical planning, $\mathsf{pre}(a) = \{p, not - p\}$ is contradictory. In temporal planning, $\mathsf{cond}_s(a) = \{p\}$ and $\mathsf{cond}_e(a) = \{not - p\}$ is a possible situation, though very unusual

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

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

**Fig. 2.** PDDL2.1 schema for two durative actions from the *driverlog* domain.

## 3 One-Shot Learning of Temporal Action Models

### 3.1 Learning Task

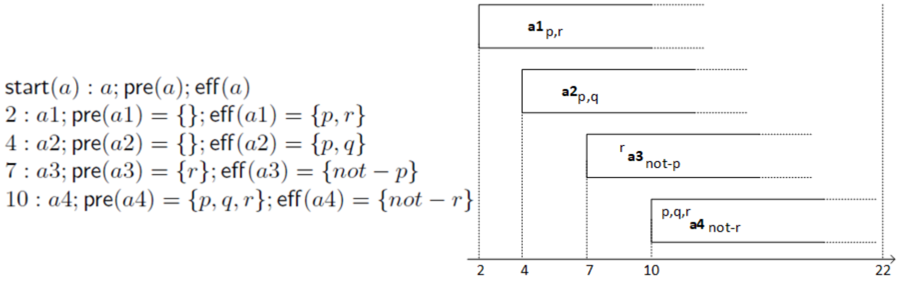
We define our one-shot learning task of a temporal action model as a tuple $\langle F, I, G, A?, O\rangle$, where:

- $\langle F, I, G, A?\rangle$ is a temporal planning problem in which actions are partially specified. Actions in $A?$ are those observed in the plan trace. They are partially specified because we do not know the exact structure in terms of distribution of conditions/effects nor the duration. In this work we assume that, for each action $a \in A?$, we only know the sets $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$, as this information can be extracted from the classical version of the planning problem, from prior knowledge we have on the problem, or given by an expert.
- $O$ is the sequence of observations corresponding to a plan trace which contains the time when every action $a$ in $A?$ starts, i.e. all $\mathsf{start}(a)$ that have been observed (by a sensor or human observer).

A solution to this learning task is a fully specified model of temporal actions $\mathcal{A}$, with all actions of $A?$, where the duration and distribution of conditions/effects is completely specified. In other words, for each action $a \in A?$, we have its equivalent version in $\mathcal{A}$ where we have learned $\mathsf{dur}(a)$, $\mathsf{cond}_s(a)$, $\mathsf{cond}_o(a)$, $\mathsf{cond}_e(a)$, $\mathsf{eff}_s(a)$ and $\mathsf{eff}_e(a)$. Actions in $\mathcal{A}$ must be consistent with the partial specification given in $A?$, having exactly the same conditions and effects, starting as observed in $O$, and inducing a temporal plan from $I$ that satisfies $G$. Intuitively, $\mathcal{A}$ is a solution to the learning task if it explains all the observations (completeness) and its subjacent temporal model implies no contradictions in the states induced by their execution (correctness).

### 3.2 Example. Is Learning a Simple Task?

Given a partially specified model of actions $A?$ and a set of observations $O$, learning a temporal action model $\mathcal{A}$ may seem, a priori, a straightforward task as it *just* implies to distribute the conditions+effects in time and estimate durations. However, this is untrue.



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

**Fig. 3.** A simple example of how learning a temporal action model from $O$ and $A?$ is not straightforward. We (optionally) observe the plan makespan is 22.

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

$a4$ needs $p$, which means two possible causal links ($\langle a1, p, a4\rangle$ or $\langle a2, p, a4\rangle$). The real causal link will be the last to happen, and this depends on the effects+durations of $a1$ and $a2$. Therefore, the causal links are unknown, not easy to detect and they affect the structure of the temporal plan. But $a4$ really needs both $a1$ and $a2$ to have $p, q, r$ supported. Let us imagine that $p, q, r$ are in $\mathsf{cond}_s(a4)$ and $p, q$ in $\mathsf{eff}_e(a2)$; then $\mathsf{dur}(a2) \leq 6$. Even if we knew for sure that $\mathsf{dur}(a2) = 6$ and $r$ was in $\mathsf{eff}_e(a1)$, we could never estimate the exact value of $\mathsf{dur}(a1)$, as any value in $]0..8]$ would be valid. Intuitively, an action has to wait until the last of its supports, but we cannot grant when the previous supports happen.Therefore, in some situations the precise duration cannot be found and we can only provide values that make the model consistent.

On the other hand, $a3$ deletes $p$, which means that it might *threat* the causal link $\langle a1, p, a4\rangle$ or $\langle a2, p, a4\rangle$. But again, this threat depends on the distribution of conditions+effects and the durations. For instance, if $not - p$ is in $\mathsf{eff}_s(a3)$, then $a1$ or $a2$ must support $p$ after time 7 and before $a4$ requires it, which entails many consistent alternatives. On the contrary, if $p$ is in both $\mathsf{eff}_s(a1)$ and $\mathsf{eff}_s(a2)$, the observations on this plan trace are inconsistent as $a3$ deletes

$p$ and no other action in the plan supports $p$ for $a4$. However, if $not - p$ is in $\mathsf{eff}_e(a3)$, $\mathsf{dur}(a3) > 3$ and $p$ is in $\mathsf{cond}_s(a4)$, then no threat will occur in the plan. Therefore, causal links and threats can easily appear or disappear depending on the selected distributions and durations.

Finally, there are some philosophical questions without a clearly motivated answer. First, why some conditions are modeled as *at start* and others as *over all*? In `drive-truck` of Fig. 2, why `(driving ?d ?t)` is required throughout the entire action but `(link ?from ?to)` only at its beginning? Apparently, the link between the two locations should remain all over the driving. So is this a wrong decision of the human modeler? Second, why some effects are modeled as *at start* and others as *at end*? In `board-truck`, why is `(not (empty ?t))` happening at start and `(driving ?d ?t)` at end? Could it be in the opposite way? Third, what happens if one action requires/supports what it deletes (see $a4$ in Fig. 3, which might threat itself)? In such a case, the delete effect should happen later than its requirement/supporting. Four, what happens if all effects are *at start*? This makes little sense, as the duration of the actions would be undetermined and could potentially exceed the known plan horizon or makespan no matter the problem goals. In Fig. 3, if the effects of $a1$ and $a2$ are *at start*, is it sensible to allow their durations to pass a hypothetical limit of 22? In other words, once all plan goals are achieved, can the actions be executed beyond the plan makespan or do they need to be cut off to such a value? This could potentially lead to an infinite number of models and overlapping situations, so it is not commonly accepted.

As can be noticed, learning a temporal action model is not simple, and many possible combinations are feasible provided they fit the constraints the model imposes. Therefore, formulating a CSP seems a promising approach to address this learning task.

## 4 A CP Formulation to Learn Temporal Planning Models

Our approach is to create a CSP that includes all the constraints the learning task requires. This includes: i) the observations on the start times; ii) the actions' conditions, effects and durations; iii) the causal structure of the plan with all possible supports; and iv) mechanisms to avoid threats and possible contradictory effects. This formulation, inspired in the work by [7], is solver-independent. This means that any off-the-shelf CSP solver that supports the expressiveness of our formulation, with binary and non-binary constraints, can be used.

### 4.1 The Variables

For each action $a$ in $A?$, we create the seven kinds of variables specified in Table 1. Variables define the time-stamps for actions, the causal links, the interval when conditions must hold and the time when the effects happen. For simplicity, and to deal with integer variables, we model time in $\mathbb{Z}^+$. To prevent time from exceeding the plan horizon, we bound all times to the makespan of the plan.[3]

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

**Table 1.** Formulation of variables and their domains for actions in $A$?.

Our temporal model formulation is more expressive than PDDL2.1 (see more details in section 4.3), and allows conditions and effects to be at any time, even outside the execution of the action. For instance, let us imagine a condition $p$ that only needs to be maintained for 5 time units before an action $a$ starts (e.g. warming-up a motor before driving): the expression $\mathsf{req\_end}(p, a) = \mathsf{start}(a); \mathsf{req\_end}(p, a) = \mathsf{req\_start}(p, a) + 5$ is possible in our formulation. Additionally, we can represent an effect $p$ that happens in the middle of action $a$: $\mathsf{time}(p, a) = \mathsf{start}(a) + (\mathsf{dur}(a)/2)$ is also possible.

Additionally, we create two dummy actions $\mathsf{init}$ and $\mathsf{goal}$ for each planning problem $\langle F, I, G, A\rangle$. First, $\mathsf{init}$ represents the initial state $I$ ($\mathsf{start}(\mathsf{init}) = 0$ and $\mathsf{dur}(\mathsf{init}) = 0$). $\mathsf{init}$ has no variables $\mathsf{sup}, \mathsf{req\_start}$ and $\mathsf{req\_end}$ because it has no conditions. $\mathsf{init}$ has as many $\mathsf{time}(p_i, \mathsf{init}) = 0$ as $p_i$ in $I$. Second, $\mathsf{goal}$ represents $G$ ($\mathsf{start}(\mathsf{goal}) = \mathit{makespan}$ and $\mathsf{dur}(\mathsf{goal}) = 0$). $\mathsf{goal}$ has as many $\mathsf{sup}(p_i, \mathsf{goal})$ and $\mathsf{req\_start}(p_i, \mathsf{goal}) = \mathsf{req\_end}(p_i, \mathsf{goal}) = \mathit{makespan}$ as $p_i$ in $G$. $\mathsf{goal}$ has no variables $\mathsf{time}$ as it has no effects.

### 4.2 The Constraints

Table 2 shows the constraints that we define among the variables of Table 1. The three first constraints are intuitive enough. The fourth constraint models the causal links. Note that in a causal link $\langle b_i, p, a\rangle$, $\mathsf{time}(p, b_i) < \mathsf{req\_start}(p, a)$ and not $\leq$. This is because temporal planning assumes an $\epsilon > 0$ as a small tolerance between the time when an effect $p$ is supported and when it is required [6]. Since we model time in $\mathbb{Z}^+$, $\epsilon = 1$ and $\leq$ becomes $<$. The fifth constraint avoids any threat via promotion or demotion [10]. The sixth constraint models the

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

fact the same action requires and deletes $p$. Note the $\geq$ inequality here; this is possible because if one condition and one effect of $a$ happen at the same time, the underlying semantics in planning considers the condition is checked instantly before the effect [6]. The seventh constraint solves the fact that two (possibly equal) actions have contradictory effects. It is important to note that these constraints involve any type of action, including $\mathsf{init}$ and $\mathsf{goal}$.

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

**Table 2.** Formulation of constraints.

### 4.3 Specific Constraints for Durative Actions of PDDL2.1

As section 2.2 explains, PDDL2.1 restricts the expressiveness of temporal planning in terms of conditions, effects, durations and structure of the actions. Hence, our temporal formulation subsumes and is significantly richer than PDDL2.1; but adding constraints to make it fully PDDL2.1-compliant is straightforward.

First, adding $((\mathsf{req\_start}(p, a) = \mathsf{start}(a))$ OR $(\mathsf{req\_start}(p, a) = \mathsf{end}(a)))$ AND $((\mathsf{req\_end}(p, a) = \mathsf{start}(a))$ OR $(\mathsf{req\_end}(p, a) = \mathsf{end}(a)))$ limits condition $p$ to be *at start*, *over all* or *at end*, i.e. $p$ is in $\mathsf{cond}_s(a)$, $\mathsf{cond}_o(a)$ or $\mathsf{cond}_e(a)$, respectively. Further, if a condition is never deleted in a plan, it can be considered as an invariant condition for such a plan. In other words, it represents static information. This type of condition is commonly used in planning to ease the grounding process from the operators; e.g. to model that there is a link between two locations and, consequently, a driving is possible, or modeling a petrol station that allows a refuel action in a given location, etc. Therefore, the constraint to be added for an invariant condition $p$ is simply: $((\mathsf{req\_start}(p, a) = \mathsf{start}(a))$ AND $(\mathsf{req\_end}(p, a) = \mathsf{end}(a)))$, i.e. $p \in \mathsf{cond}_o(a)$. Surprisingly, invariant conditions are modeled differently depending on the human modeler. See, for instance, `(link`

`?from ?to)` of Fig. 2, which is modeled as an *at start* condition despite: i) the link should be necessary all over the driving, and ii) no action in this domain can be planned to delete that link. This also happens in the *transport* domain of the IPC, where a refuel action requires to have a petrol station in a location only *at start*, rather than *over all* which makes more sense. This shows that modeling a planning domain is not easy and it highly depends on the human's decision. On the contrary, our formulation checks the invariant conditions and deals with them always in the same coherent way.

Second, $((\mathsf{time}(p,a) = \mathsf{start}(a))$ OR $(\mathsf{time}(p,a) = \mathsf{end}(a)))$ makes an effect $p$ happen only *at start* or *at end* of action $a$, i.e. $p$ is in $\mathsf{eff}_s(a)$ or $\mathsf{eff}_e(a)$. Also, if all effects happen *at start* the duration of the action would be irrelevant and could exceed the plan makespan. To avoid this, for any action $a$, at least one of its effects should happen *at end*: $\sum_{i=1}^{n=|\mathsf{eff}(a)|} \mathsf{time}(p_i, a) > n \cdot \mathsf{start}(a)$, which guarantees $\mathsf{eff}_e(a)$ is not empty.

Third, durations in PDDL2.1 can be defined in two different ways. On the one hand, durations can be equal for all grounded actions of the same operator. For instance, any instantiation of `board-truck` of Fig. 2 will last 2 time units no matter its parameters. Although this may seem a bit odd, it is not an uncommon practice to simplify the model. The constraint to model this is: $\forall a_i, a_j$ being instances of the same operator: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$. On the other hand, although different instantiations of `drive-truck` will last different depending on the locations, different occurrences of the same instantiated action will last equal. In a PDDL2.1 temporal plan, multiple occurrences of `drive-truck(truck1,loc1,loc2,driver1)` will have the same duration no matter when they start. Intuitively, they are different occurrences of the same action, but in the real-world the durations would differ from driving at night or in peak times. Since PDDL2.1 makes no distinction among different occurrences, the constraint to add is: $\forall a_i, a_j$ being occurrences of the same durative action: $\mathsf{dur}(a_i) = \mathsf{dur}(a_j)$. Obviously, this second constraint is subsumed by the first one in the general case where all instances of the same operator have the same duration.

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

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

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

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

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

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

### 4.4 Example

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

The variables and domains are: $\mathsf{start}(a3) = 7$; $\mathsf{dur}(a3) \in [1..15]$; $\mathsf{end}(a3) = \mathsf{start}(a3) + \mathsf{dur}(a3)$; $\mathsf{sup}(r, a3) \in \{a1\}$; $\mathsf{req\_start}(r, a3), \mathsf{req\_end}(r, a3) \in [0..22]$; and $\mathsf{time}(not - p, a3) \in [0..22]$. On the other hand, the constraints are: $\mathsf{end}(a3) \leq \mathsf{start}(\mathsf{goal})$; $\mathsf{req\_start}(r, a3) \leq \mathsf{req\_end}(r, a3)$; if $\mathsf{sup}(r, a3) = a1$ then $\mathsf{time}(r, a1) < \mathsf{req\_start}(r, a3)$; if $\mathsf{sup}(r, a3) = a1$ then $((\mathsf{time}(not - r, a4) < \mathsf{time}(r, a1))$ OR $(\mathsf{time}(not{-}r, a4) > \mathsf{req\_end}(r, a3)))$; $\mathsf{time}(not{-}p, a3) \neq \mathsf{time}(p, a1)$ and $\mathsf{time}(not{-}p, a3) \neq \mathsf{time}(p, a2)$.

There are many consistent solutions for this simple example, mainly because there is a huge range of possible durations that make the learned model consistent with the partially specified model $A?$. Fig. 4 shows six arbitrary models as solutions. What is important to note is that the structure, i.e. distribution of conditions/effects, is similar in all the learned models. Actually, the distribution of the effects is identical (except for $q$ in model 2), and the distribution of conditions is very similar (e.g. $q$ is always in $\mathsf{cond}_o$ and $r$ in $a4$ is very often in $\mathsf{cond}_o$). This shows that the one-shot learning returns not only consistent models but also similar, which is very positive. The durations are, however, more different: $\mathsf{dur}(a1)$ ranges in these models from 7 to 19, whereas $\mathsf{dur}(a2)$ ranges from 5 to 18. As explained in section 3.2, learning the precise duration from just one sample may not be always possible, which is the main limitation of the one-shot learning task. In fact, the specific constraint of PDDL2.1, with regard to having multiple occurrences of the same action having the same duration, can significantly help us to learn the actions' duration in a more precise way as the learned duration must be consistent with all those occurrences.

### 4.5 Implementation. Use of Heuristics for Resolution

Our CSP formulation is automatically compiled from a partially specified action model, as defined in a classical planning problem, and the observations from a plan execution. The formulation has been implemented in Choco[4], an open-source Java library for constraint programming that provides an object-oriented API to state the constraints to be satisfied.

Our formulation is solver-independent, which means we do not use heuristics that may require changes in the implementation of the CSP engine. Although this reduces the solver performance, we are interested in using it as a blackbox that can be easily changed with no modification in our formulation. However,

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

| Action | $\mathsf{dur}$ | $\mathsf{cond}_s$ | $\mathsf{cond}_o$ | $\mathsf{cond}_e$ | $\mathsf{eff}_s$ | $\mathsf{eff}_e$ |
|---|---|---|---|---|---|---|
| Learned model 1 | | | | | | |
| $a1$ | 8 | | | | $r$ | $p$ |
| $a2$ | 18 | | | | $q$ | $p$ |
| $a3$ | 1 | | $r$ | | | $not-p$ |
| $a4$ | 1 | | $q, r$ | $p$ | | $not-r$ |
| Learned model 2 | | | | | | |
| $a1$ | 19 | | | | $r$ | $p$ |
| $a2$ | 5 | | | | | $p, q$ |
| $a3$ | 1 | | | $r$ | | $not-p$ |
| $a4$ | 1 | $r$ | $p, q$ | | | $not-r$ |
| Learned model 3 | | | | | | |
| $a1$ | 7 | | | | $r$ | $p$ |
| $a2$ | 18 | | | | $q$ | $p$ |
| $a3$ | 1 | | | $r$ | | $not-p$ |
| $a4$ | 1 | | $p, q, r$ | | | $not-r$ |

| Action | $\mathsf{dur}$ | $\mathsf{cond}_s$ | $\mathsf{cond}_o$ | $\mathsf{cond}_e$ | $\mathsf{eff}_s$ | $\mathsf{eff}_e$ |
|---|---|---|---|---|---|---|
| Learned model 4 | | | | | | |
| $a1$ | 7 | | | | $r$ | $p$ |
| $a2$ | 9 | | | | $q$ | $p$ |
| $a3$ | 1 | | | $r$ | | $not-p$ |
| $a4$ | 1 | | $p, q, r$ | | | $not-r$ |
| Learned model 5 | | | | | | |
| $a1$ | 9 | | | | $r$ | $p$ |
| $a2$ | 6 | | | | $q$ | $p$ |
| $a3$ | 1 | | $r$ | | | $not-p$ |
| $a4$ | 1 | | $q, r$ | $p$ | | $not-r$ |
| Learned model 6 | | | | | | |
| $a1$ | 8 | | | | $r$ | $p$ |
| $a2$ | 16 | | | | $q$ | $p$ |
| $a3$ | 1 | $r$ | | | | $not-p$ |
| $a4$ | 1 | | $q, r$ | $p$ | | $not-r$ |

**Fig. 4.** Six learned models for the example of Fig. 3, but there are many more.

we can easily encode standard static heuristics for variable and value selection that help improve efficiency by following the next ordering, which has shown very efficient in our experiments:

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

### 4.6 Using the CP Formulation for Plan Validation

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

In particular, the possibility of adding those constraints is very appealing when used for validating whether a partial action model allows us to learn a consistent model, as we will see in section 5. Let us assume that the distribution of all (or just a few) conditions and/or effects is known and, in consequence, represented in the learning task. If a solution is found, then that structure of conditions/effects is consistent for the learned model. On the contrary, if no solution is found that structure is inconsistent and cannot be explained. Analogously, we can represent known values for the durations. If a solution is found, the durations are consistent, and inconsistent otherwise. Hence, we have three options for validating a partial model *w.r.t.*: i) a known structure with the distribution of conditions/effects; ii) a known set of durations; and iii) a known structure plus a known set of durations (i+ii). The first and second option allows for some flexibility in the learning task because some variables remain open. On the contrary, the third option checks whether a learned model can fit the given constraints, thus reproducing a strict plan validation task equivalent to [11].

## 5 Evaluation

### 5.1 Evaluation Metrics

The empirical evaluation of a learning task can be addressed from two perspectives. From a pure syntactic perspective, learning can be considered as an automated design task to create a new model that is similar to a reference (or *ground truth*) model. Consequently, the success of learning is an accuracy measure of how similar these two models are, which usually counts the number of differences (in terms of incorrect durations or distribution of conditions/effects). Unfortunately, there is not a unique reference model when learning temporal models at real-world problems. Also, a pure syntax-based measure usually returns misleading and pessimistic results, as it may count as incorrect a different duration or a change in the distribution of conditions/effects that really represent equivalent reformulations of the reference model. For instance, given the example of Fig. 2, the condition learned `(over all (link ?from ?to))` would be counted as a difference in action `drive-truck`, as it is `at start` in the reference model; but it is, semantically speaking, even more correct. Analogously, some durations may differ from the reference model but they should not be counted as incorrect. As seen in section 3.2, some learned durations cannot be granted, but the underlying model is still consistent. Therefore, performing a syntactic evaluation in learning is not always a good idea.

From a semantic perspective, learning can be considered as a classification task where we first learn a model from a training dataset, then tune the model on a validation test and, finally, asses the model on a test dataset. Our approach represents a one-shot learning task because we only use one plan sample to learn the model and no validation step is required. Therefore, the success of the learned model can be assessed by analyzing the success ratio of the learned model *vs.* all the unseen samples of a test dataset. In other words, we are interested in learning a model that fits as many samples of the test dataset as possible. This is the evaluation that we consider most valuable for learning, and define the success ratio as the percentage of samples of the test dataset that are consistent with the learned model. A higher ratio means that the learned model explains, or adequately fits, the observed constraints the test dataset imposes.

### 5.2 Experimental Results

We have run experiments on nine IPC planning domains. It is important to highlight that these domains are encoded in PDDL2.1, with the number of operators shown in Table 3, so we have included the constraints given in section 4.3. We first get the plans for these domains by using five planners (*LPG-Quality* [9], *LPG-Speed* [9], *TP* [12], *TFD* [5] and *TFLAP* [16]), where the planning time is limited to 100s. The actions and observations on each plan are automatically compiled into a CSP learning instance. Then, we run the one-shot learning task to get a temporal action model for each instance, where the learning time is limited to 100s on an Intel i5-6400 @ 2.70GHz with 8GB of RAM. In order to assess the quality of the learned model, we validate each model *vs.* the other models *w.r.t.* the *struct*ure, the *dur*ation and the *struct*ure+*dur*ation, as discussed in section 4.6. For instance, the *zenotravel* domain contains 78 instances, which means learning 78 models. Each model is validated by using the 77 remaining models, thus producing 78×77=6006 validations per struct, dur and struct+dur each. The value for each cell is the average success ratio. In *zenotravel*, the struct value means that the distribution of conditions/effects learned by using only one plan sample is consistent with all the samples used as dataset (100% of the 6006 validations), which is the perfect result, as also happens in *floortile* and *sokoban* domains. The dur value means the durations learned explain 68.83% of the dataset. This value is usually lower because any learned duration that leads to inconsistency in a sample counts as a failure. The struct+dur value means that the learned model explains entirely 35.76% of the samples. This value is always the lowest because a subtle structure or duration that leads to inconsistency in a sample counts as a failure. As seen in Table 3, the results are specially good, taking into consideration that we use only one sample to learn the temporal action model. These results depend on the domain size (number of operators, which need to be grounded), the relationships (causal links, threats and interferences) among the actions, and the size and quality of the plans.

We have observed that some planners return plans with unnecessary actions, which has a negative impact for learning precise durations. The worst result is returned in the *rovers* domain, which models a group of planetary rovers to explore the planet they are on. Since there are many parallel actions for taking pictures/samples and navigation of multiple rovers, learning the duration and the structure+duration is particularly complex in this domain.

## 6 Conclusions

We have presented a purely declarative CP formulation, which is independent of any CSP solver, to address the learning of temporal action models. Learning in planning is specially interesting to recognize past behavior in order to predict

| Domain | No. operators | No. instances | struct | dur | struct+dur |
|---|---|---|---|---|---|
| *zenotravel* | 5 | 78 | 100% | 68.83% | 35.76% |
| *driverlog* | 6 | 73 | 97.60% | 44.86% | 21.04% |
| *depots* | 5 | 64 | 55.41% | 76.22% | 23.19% |
| *rovers* | 9 | 84 | 78.84% | 5.35% | 0.17% |
| *satellite* | 5 | 84 | 80.74% | 57.13% | 40.53% |
| *storage* | 5 | 69 | 58.08% | 70.10% | 38.36% |
| *floortile* | 7 | 17 | 100% | 80.88% | 48.90% |
| *parking* | 4 | 49 | 86.69% | 81.38% | 54.89% |
| *sokoban* | 3 | 51 | 100% | 87.25% | 79.96% |

**Table 3.** Average success ratio of the one-shot learned model *vs.* the test dataset in different IPC planning domains.

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

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

Finally, our CP formulation can be represented and solved by Satisfiability Modulo Theories, which is part of our current work. As future work, we want to extend our formulation to learn meta-models, as combinations of many learned models, and a more complete action model. In the latter, rather than using a partially specified set of actions, we want to find out the conditions/effects together with their distribution. The underlying idea of finding an action model consistent with all the constraints will remain the same, but the model will need to be extended with additional decision variables and constraints. This will probably lead to the analysis of new heuristics for resolution.

## References

1. Amir, E., Chang, A.: Learning partially observable deterministic action models. Journal of Artificial Intelligence Research **33**, 349–402 (2008)
2. Arora, A., Fiorino, H., Pellier, D., Métivier, M., Pesty, S.: A review of learning planning action models. The Knowledge Engineering Review **33** (2018)
3. Cresswell, S.N., McCluskey, T.L., West, M.M.: Acquiring planning domain models using LOCM. The Knowledge Engineering Review **28**(02), 195–213 (2013)
4. Cushing, W., Kambhampati, S., Weld, D.S., et al.: When is temporal planning really temporal? In: Proceedings of the 20th international joint conference on Artifical intelligence. pp. 1852–1859. Morgan Kaufmann Publishers Inc. (2007)
5. Eyerich, P., Mattmüller, R., Röger, G.: Using the context-enhanced additive heuristic for temporal and numeric planning. In: Nineteenth International Conference on Automated Planning and Scheduling (2009)
6. Fox, M., Long, D.: Pddl2.1: An extension to pddl for expressing temporal planning domains. Journal of artificial intelligence research **20**, 61–124 (2003)
7. Garrido, A., Arangu, M., Onaindia, E.: A constraint programming formulation for planning: from plan scheduling to plan generation. Journal of Scheduling **12**(3), 227–256 (2009)
8. Geffner, H., Bonet, B.: A concise introduction to models and methods for automated planning. Synthesis Lectures on Artificial Intelligence and Machine Learning **8**(1), 1–141 (2013)
9. Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs in lpg. Journal of Artificial Intelligence Research **20**, 239–290 (2003)
10. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: theory and practice. Elsevier (2004)
11. Howey, R., Long, D., Fox, M.: Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In: 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004). pp. 294–301 (2004)
12. Jiménez, S., Jonsson, A., Palacios, H.: Temporal planning with required concurrency using classical planning. In: Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS) (2015)
13. Jiménez, S., De la Rosa, T., Fernández, S., Fernández, F., Borrajo, D.: A review of machine learning for automated planning. The Knowledge Engineering Review **27**(4), 433–467 (2012)
14. Kambhampati, S.: Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In: Proceedings of the National Conference on Artificial Intelligence (AAAI-07). vol. 22(2), pp. 1601–1604 (2007)
15. Kucera, J., Barták, R.: LOUGA: learning planning operators using genetic algorithms. In: Pacific Rim Knowledge Acquisition Workshop, PKAW-18. pp. 124–138 (2018)
16. Marzal, E., Sebastia, L., Onaindia, E.: Temporal landmark graphs for solving overconstrained planning problems. Knowledge-Based Systems **106**, 14–25 (2016)
17. Mourão, K., Zettlemoyer, L.S., Petrick, R.P.A., Steedman, M.: Learning STRIPS operators from noisy and incomplete observations. In: Conference on Uncertainty in Artificial Intelligence, UAI-12. pp. 614–623 (2012)
18. Yang, Q., Wu, K., Jiang, Y.: Learning action models from plan examples using weighted MAX-SAT. Artificial Intelligence **171**(2-3), 107–143 (2007)

19. Zhuo, H.H., Kambhampati, S.: Action-model acquisition from noisy plan traces. In: International Joint Conference on Artificial Intelligence, IJCAI-13. pp. 2444–2450 (2013)
20. Zhuo, H.H., Kambhampati, S.: Model-lite planning: Case-based vs. model-based approaches. Artificial Intelligence **246**, 1–21 (2017)
21. Zhuo, H.H., Nguyen, T.A., Kambhampati, S.: Refining incomplete planning domain models through plan traces. In: International Joint Conference on Artificial Intelligence, IJCAI-13. pp. 2451–2458 (2013)
22. Zhuo, H.H., Yang, Q., Hu, D.H., Li, L.: Learning complex action models with quantifiers and logical implications. Artificial Intelligence **174**(18), 1540–1569 (2010)