Improving the Expressiveness of Planning Models with State Observations and Constraint Programming

Antonio Garrido and **Sergio Jiménez**
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València.
Camino de Vera s/n. 46022 Valencia, Spain
{agarridot,serjice}@dsic.upv.es

Abstract

This paper shows that existing CSP compilations for temporal planning can be adapted to improve the expressiveness of a given classical planning model with regard to observations of plan executions.

Introduction

Automated Planning is the model-based approach for the task of selecting actions that achieve a given set of goals. *Classical planning* is the vanilla model for automated planning. This planning model assumes: fully observable states, actions with deterministic and instant effects and, goals that are exclusively referred to the last state reached by a plan (Geffner and Bonet 2013).

Besides *classical planning*, there is a bunch of more expressive planning models that take these aspects into account to compute more detailed solutions than classical plans. One of these models is *temporal planning*, that relaxes the assumption of instant effects to compute plans that indicate the precise time-stamp where actions start and end. Actions in *temporal planning* can then have durations, be applied in parallel and overlap (Ghallab, Nau, and Traverso 2004).

Despite the potential of automated planning (state-of-the-art planners are able to compute plans with hundreds of actions in seconds time), its applicability is still limited because of the complexity of specifying correct and complete planning models. The more expressive the planning model, the more evident becomes this knowledge acquisition bottleneck. In this paper we show that existing CSP compilations for temporal planning can be adapted for improving the expressiveness of a given classical planning model with regard to observations of plan executions.

Background

This section formalizes the models of *classical* and *temporal planning* that we use in the paper as well as the *Constraint Satisfaction Problem*, the techinque we leverage to improve the expressiveness of a given classical planning model with regard to observations of plan executions.

Classical Planning

We use F to denote the set of *fluents* (propositional variables) describing a state. A *literal* l is a valuation of a fluent $f \in F$; i.e. either $l = f$ or $l = \neg f$. A set of literals L represents a partial assignment of values to fluents (without loss of generality, we will assume that L does not contain conflicting values). We use $\mathcal{L}(F)$ to denote the set of all literal sets on F; i.e. all partial assignments of values to fluents. A *state* s is a full assignment of values to fluents; $|s| = |F|$.

A classical planning action a is defined as:

- $\mathsf{pre}(a) \in \mathcal{L}(F)$, the *preconditions* of a, is the set of literals that must hold for the action $a \in A$ to be applicable.
- $\mathsf{eff}^+(a) \in \mathcal{L}(F)$, the *positive effects* of a, is the set of literals that become true after the application of the action $a \in A$.
- $\mathsf{eff}^-(a) \in \mathcal{L}(F)$, the *negative effects* of a, is the set of literals that become false after the application of the action.

We say that an action $a \in A$ is *applicable* in a state s iff $\mathsf{pre}(a) \subseteq s$. The result of applying a in s is the *successor state* denoted by $\theta(s,a) = \{s \setminus \mathsf{eff}^-(a)) \cup \mathsf{eff}^+(a)\}$.

We assume that actions $a \in A$ are instantiated from given action schemas, as in PDDL. Figure 1 shows the *fly* action schema from the *zenotravel* domain encoded in PDDL2.1 (Fox and Long 2003). According to this action model an aircraft moves from one city to another consuming a single unit of fuel.

```
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city ?l1 ?l2 - flevel)
 :precondition (and (at ?a ?c1) (fuel-level ?a ?l1)
                    (next ?l2 ?l1))
 :effect (and (not (at ?a ?c1)) (at ?a ?c2)
              (not (fuel-level ?a ?l1))
              (fuel-level ?a ?l2)))
```

Figure 1: PDDL1.1 encoding of the classical action model of the *fly* operator from the *zenotravel* domain.

A *classical planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where I is an initial state and $G \in \mathcal{L}(F)$ is a goal condition. A *sequential plan* is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$ whose execution induces the *state trajectory* $s = \langle s_0, s_1, \ldots, s_n\rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq$

n, a_i is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$. The *plan length* is denoted with $|\pi| = n$. A plan π *solves* P iff $G \subseteq s_n$, i.e., if the goal condition is satisfied at the last state reached after following the application of the plan π in the initial state I. A solution plan for P is *optimal* if it has minimum length.

Temporal Planning

A *temporal planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where the set of fluents F, the initial state i, and the goal conditions $G \subseteq \mathcal{L}(F)$ are defined as for a *classical planning problem*. However, A represents here a set of *temporal actions* such that each $a \in A$ is defined as follows:

- $d(a)$, the action duration.
- $\mathsf{cond}_s(a) \subseteq \mathcal{L}(F)$, $\mathsf{cond}_o(a) \subseteq \mathcal{L}(F)$ and $\mathsf{cond}_e(a) \subseteq \mathcal{L}(F)$, that respectively represent the *conditions* (literals that must hold for the action a to be applicable) *at start*, *over all*, and *at end* of the action.
- $\mathsf{add}_s(a) \subseteq \mathcal{L}(F)$ and $\mathsf{add}_e(a) \subseteq \mathcal{L}(F)$, that are the *positive effects* (fluents set to true by the application of a) *at start* and *at end* of the action application.
- $\mathsf{del}_s(a) \subseteq \mathcal{L}(F)$ and $\mathsf{del}_e(a) \subseteq \mathcal{L}(F)$, that are the *negative effects* (also *at start* and *at end*).

Despite temporal actions have a duration, conditions *at start* and *at end* are checked instantaneously. In the same way, *at start* and *at end* effects are instantaneously applied (effects can only happen *at start* or *at end* since continuous effects are not considered). With this regard, the semantics of a temporal action a can be defined in terms of two discrete events $start_a$ and end_a. The duration imposes that end_a must occur exactly $d(a)$ time units after $start_a$ and *over all* conditions of a must hold at any state between $start_a$ and end_a.

As an example, Figure 2 shows the action model of the *fly* operator from the *zenotravel* domain encoded in PDDL2.1 (Fox and Long 2003).

```
(:durative-action fly
 :parameters (?a - aircraft ?c1 ?c2 - city ?l1 ?l2 - flevel)
 :duration (= ?duration 180)
 :condition (and (at start (at ?a ?c1))
                 (at start (fuel-level ?a ?l1))
                 (at start (next ?l2 ?l1)))
 :effect (and (at start (not (at ?a ?c1)))
              (at end (at ?a ?c2))
              (at end (not (fuel-level ?a ?l1)))
              (at end (fuel-level ?a ?l2))))
```

Figure 2: PDDL2.1 encoding of the temporal action *fly* from the *zenotravel* domain.

A *temporal plan* is a set of pairs $\pi = \langle (a_1, t_1), \ldots, (a_n, t_n)\rangle$ such that each pair contains a temporal action $a \in A$ and its scheduled start time. Note that each $(a, t_a) \in \pi$ pair induces two discrete events, $start_a$ and end_a with associated time-stamps t and $t + d(a)$. If we order the $2n$ events induced from π by their associated times, we obtain the *event sequence*, $E_\pi = \langle e_1, \ldots, e_m\rangle$, where $1 \leq m \leq 2n$. Each e_i, $1 \leq i \leq m$, is a *joint event* composed of one or more individual events of π that all have the same associated time. We say that π has *simultaneous events* if $m < 2n$, i.e. if at least one joint event is composed of multiple individual events.

The *execution* of a temporal plan π starting from I, induces the *state trajectory* $\tau = \langle s_0, s_1, \ldots, s_m\rangle$ such that $s_0 = I$ and a_i $(1 \leq i \leq m)$ is a classical planning action modeling the corresponding *joint event* e_i and satisfying that a_i is applicable in s_{i-1} and that the application of a_i generates the successor state $s_i = \theta(s_{i-1}, a_i)$ (Jiménez, Jonsson, and Palacios 2015). A *temporal plan* π is a solution for P iff the last state reached by its execution starting from I satisfies that $G \subseteq s_m$.

The quality of a temporal plan is given by its *makespan*, i.e. the temporal duration from the the start of the first temporal action to the end of the last temporal action. Without loss of generality, we assume that the first temporal action is scheduled to start at time 0, i.e. $min_{\langle a, t_a\rangle \in \pi} = 0$. In this case, the makespan of a temporal plan π is formally defined as $max_{\langle a, t_a\rangle \in \pi}(t_a + d(a))$

The observation model

In this work we assume that there is *partial observability* of the execution of temporal plans and that observations are *noiseless*, meaning that if the value of a fluent is observed, then the observation is correct.

Given a *temporal planning problem* $P = \langle F, A, I, G\rangle$, and a temporal plan π s.t. π solves P and it induces the state trajectory $\tau = \langle s_0, s_1, \ldots, s_m\rangle$. The *observation* of that trajectory is denoted by $obs(\tau)$ and it defines the same sequence of states induced by the execution of π on P but where the value of certain fluents may be omitted, i.e. $|s^o| \leq |F|$ for every $s^o \in obs(\tau)$. Each observed state is labeled with a time-stamp.

Definition 1 (Explaning an observation)**.** *Given a* temporal planning problem P *and a sequence of partially observed states* $\mathcal{O}(\tau)$*, we say that a plan* π explains the observation *(denoted* $\pi \mapsto \mathcal{O}(\tau)$*) iff* π *is a solution for* P *that is consistent with* $\mathcal{O}(\tau)$*. If* π *is also optimal, we say that* π *is the* best explanation *for* $\mathcal{O}(\tau)$.

Given a temporal planning problem $P = \langle F, A, I, G\rangle$, we say that an action model $\mathcal{M}$ is a definition of the $\langle \rho, \theta\rangle$ functions of every action in A. Further we say that a model $\mathcal{M}$ *explains* a sequence of observations $\mathcal{O}(\tau)$ iff, when the $\langle \rho, \theta\rangle$ functions of the actions in P are given by $\mathcal{M}$, there exists a solution plan for P that explains $\mathcal{O}(\tau)$.

Constraint Satisfaction Problem

A *Constraint Satisfaction Problem* is defined as the triple $\langle X, D, C\rangle$ where:

- $X = \langle x_0, \ldots, x_n\rangle$ is a set of n finite domain variables.
- $D = \langle D_{x_0}, \ldots, D_{x_n}\rangle$ are the respective domains defining the set of possible values for each variable $x \in X$.
- $C = \langle c_0, \ldots, c_m\rangle$ is a set of constraints bounding the possible values of the variables in X. Every constraint $c \in C$, is in turn a pair $c = (X_c, r_c)$ where:
  - $X_c \subseteq X$ is a subset of k variables.

- r_c is a k-ary relation on the corresponding subset of domains.

An *evaluation* of the variables is a function from a subset of variables to a particular set of values in the corresponding subset of domains. An evaluation v *satisfies* a constraint $c = (X_c, r_c)$ if the values assigned to the variables in X_c satisfy the relation r_c. An evaluation of X is *consistent* if it does not violate any of the constraints in C. An evaluation is *complete* if it includes values for all the variables. An evaluation is a *solution* for a given CSP $\langle X, D, C\rangle$ if it is both *consistent* and *complete*.

Building Temporal Planning Models

Our approach for building temporal planning models from a classical planning model and observations of plan executions is to compile this task into a CSP.

The modeling task

First we formalize the task of building temporal planning models from a classical planning model and a set of observations of plan executions with the pair $\Lambda = \langle P, obs(\tau)\rangle$:

- P is a classical planning problem $P = \langle F, A, I, G\rangle$.
- $obs(\tau)$ is a sequence of partial states that corresponds to a noiseless observation of the actual temporal plan execution such that $s_0 \in obs(\tau)$ is a *fully observed* state, i.e. $|s_0| = |F|$. Consequently, the corresponding set of predicates and objects that shape the fluents in F are inferrable from s_0.

A *solution* to a learning task $\Lambda = \langle P, obs(\tau)\rangle$ a temporal action model $\mathcal{M}$ that *explains* the input observation $obs(\tau)$.

Example

Figure 3 shows an example of a temporal plan for solving a problem from the *zenotravel domain*.

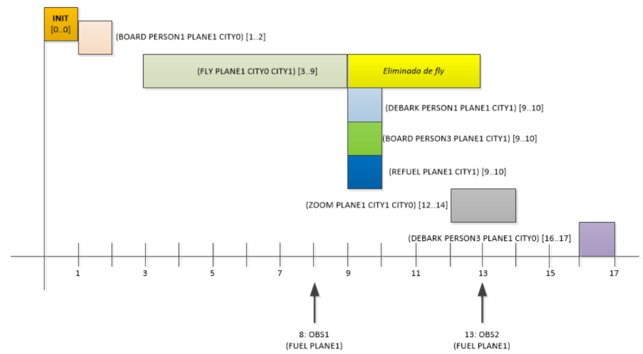

Figure 3: Example of a temporal plan for solving a problem from the *zenotravel domain*.

The compilation

Given an observation $obs(\tau)$ of the execution of a temporal plan. The CSP task that is output by our compilation is built as follows.

The set of **variables** X comprises:

- For each action a observed in a ω:
  - $start_a$, indicating the scheduled time-stamp for the start of the observed action a. The domain of this variable is $[t_a]$, i.e. a singleton given by the input observation.
  - $duration_a$ representing the action duration. The domain of this variable is $[0, max_t]$ where max_t is the maximum-time stamp in the given input observation $obs(\tau)$.
  - end_a is a derived variable, defined in $\mathbb{Z}^+$, that indicates the time-stamp for the end of the observed action a. The value of this variable is $end_a = start_a + duration_a$.
  - $preS_{f,a}$ and $preE_{f,a}$ are Boolean variables indicating whether fuent f is a condition (*at start* or *at end*) of the action a in the input. This set of variables is given by the input classical planning model.
  - $supports_{f,a_i}$ indicates that action a_j is the supporter of the fluent f for action a_i in the observed plan. The domain of this variable is the set of actions.
  - $time_{f,a}$ is a variable indicating when the value of f is modified as a result of applying some effects of action a. The domain of this variable is $[start_a, max_t]$.

The domains of the variables in X is bound by the following set of **constraints**:

- For each action a observed in a ω:
  - $end_a = start_a + duration_a$.
  - $time_{f,a_i} != time_{\neg f,a_j}$.
  - $time_{f,a} = start_a$ XOR $time_{f,a} = end_a$.
  - $start_a \leq preS_{f,a} \leq preE_{f,a} \leq end_a$.
  - IF $supports_{f,a_i} = a_j$ THEN:
    1. $time_{f,a_i} < preS_{f,a_j}$.
    2. $time_{f,a_i} < end_{a_j}$.
    3. $time_{f,a_k} < time_{f,a_i}$ OR $time_{f,a_k} < time_{f,a_j}$.
  - $(start_{a_1} = preS_{f,a_1}$ AND $start_{a_n} = preS_{f,a_{a_n}})$ OR $(end_{a_1} = preS_{f,a_1}$ AND $end_{a_n} = preS_{f,a_{a_n}})$. Con-

straints of the same kind are also defined for the variables $preE_{f,a}$ and $time_{f,a}$.

– .

Given a solution for the CSP output by our compilation, the action model $\mathcal{M}$ that solves $\Lambda = \langle P, obs(\tau)\rangle$ is computable in linear time and space.

Compilation properties

Lemma 2. *Soundness. Any solution for the CSP output by our compilation induces a set of action models $\mathcal{M}'$ that solves $\Lambda = \langle \mathcal{M}, \omega\rangle$.*

Proof sketch. □

Lemma 3. *Completeness. Any set of action models $\mathcal{M}'$ that solves $\Lambda = \langle \mathcal{M}, \omega\rangle$ is computable solving the CSP output by our compilation.*

Proof sketch. □

An interesting aspect of our compilation approach is that when an input temporal planning model $\mathcal{M}$ is given in Λ, the compilation serves to validate whether the observation $obs(\tau)$ follows the given model $\mathcal{M}$:

- $\mathcal{M}$ is proved to be a *valid* action model for the given input data in $obs(\tau)$ iff a solution for the output CSP can be found.
- $\mathcal{M}$ is proved to be a *invalid* action model for the given input data $obs(\tau)$ iff the output CSP is unsolvable. This means that $\mathcal{M}$ cannot *explain* the given observation of the plan execution.

This validation capacity of our compilation is beyond the functionality of VAL (the plan validation tool (Howey, Long, and Fox 2004)) because our approach is able to address *model validation* of a partial (or even an empty) action model with a partially observed plan trace. On the other hand, VAL requires (1) a full plan and (2), a full action model for plan validation.

Related work

Boolean satisfiability (SAT) is a powerful problem solving approach that has shown successful to address challenging classical planning task (Kautz and Selman 1999; Rintanen 2009; 2012). Likewise *Constraint Satisfaction Problems* (CSP) has also been used to synthesize solution plans to numeric and temporal planning problems (Do and Kambhampati 2001; Lopez and Bacchus 2003; Vidal and Geffner 2006; Garrido, Arangu, and Onaindia 2009).

Evaluation

Conclusions

References

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into csp. *Artificial Intelligence* 132(2):151–182.

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research* 20:61–124.

Garrido, A.; Arangu, M.; and Onaindia, E. 2009. A constraint programming formulation for planning: from plan scheduling to plan generation. *Journal of Scheduling* 12(3):227–256.

Geffner, H., and Bonet, B. 2013. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(1):1–141.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, 294–301.

Jiménez, S.; Jonsson, A.; and Palacios, H. 2015. Temporal planning with required concurrency using classical planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*.

Kautz, H., and Selman, B. 1999. Unifying sat-based and graph-based planning. In *IJCAI*, volume 99, 318–325.

Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a csp. In *IJCAI*, volume 3, 954–960.

Rintanen, J. 2009. Planning and sat. *Handbook of Satisfiability* 185:483–504.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal pocl planner based on constraint programming. *Artificial Intelligence* 170(3):298.