Learning Temporal Planning Models

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

Introduction

Automated Planning is the model-based approach for the task of selecting actions that achieve a given set of goals (Geffner and Bonet 2013). The *classical planning* model is the vanilla model for automated planning and it assumes: fully observable states, actions with deterministic and instant effects and, goals that are conditions referred to the last state reached by the selected actions. *Temporal planning* models relax the assumption of instant effects so actions can have durations, be applied in parallel and overlap (Ghallab, Nau, and Traverso 2004).

Despite the potential of automated planning, its applicability is limited given the complexity of defining planning models. This knowledge acquisition bottleneck becomes more evident in the more expressive planning models, like temporal planning. In this paper we introduce a novel method to automatically build temporal models from observations of action selection episodes.

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). In this paper we show that existing CSP compilation for the synthesis of temporal plans can be adapted for learning temporal planning models from from observations of action selection episodes.

Background

This section formalizes: (1) the *temporal planning* model that we aim to learn and (2), the kind of *observations* of the execution of temporal plans that we aim to learn from and (3) *Constraint Satisfaction Problem*, the problem solving approach we follow to address the task of learning temporal models.

Temporal 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; $|s| = |F|$ with its associated time-stamp t_s. Explicitly including negative literals $\neg f$ in states simplifies subsequent definitions but often we will abuse of notation by defining a state s only in terms of the fluents that are true in s, as it is common in STRIPS planning.

A *temporal planning problem* is a tuple $P = \langle F, A, I, G \rangle$, where I is an initial state, $G \subseteq \mathcal{L}(F)$ is a goal condition, and A is a set of *temporal actions* $a \in A$ defined as follows:

- $d(a)$, the action duration.
- $\mathsf{pre}_s(a) \subseteq \mathcal{L}(F)$, $\mathsf{pre}_o(a) \subseteq \mathcal{L}(F)$ and $\mathsf{pre}_e(a) \subseteq \mathcal{L}(F)$, that respectively represent the *preconditions* (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, preconditions *at start* and *at end* are checked instantaneously. In the same way, *at start* and *at end* effects are instantaneously applied. 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* preconditions of a must hold in all states between $start_a$ and end_a.

As an example, Figure 1 shows the action model of the *fly* operator from the *zenotravel* domain encoded in PDDL.

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 $t + d(a)$.

```
(: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 1: PDDL encoding of the action model of the *fly* operator from the *zenotravel* domain.

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* 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 *make-span*, 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. $min_{(a,t_a)\in\pi} = 0$. In this case, the make-span of a temporal plan π is formally defined as $max_{(a,t_a)\in\pi}(t_a + d(a))$

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 or an action is observed, then the observation is correct. Formally given a *temporal planning problem* $P = \langle F, A, I, G \rangle$, the *observation* of the execution of a plan π that solves P is defined as a pair $\omega = \langle obs(\pi), obs(\tau) \rangle$ where:

- $obs(\pi)$ is a sub-sequence of the pairs action-scheduled time in π, where the scheduled time might be unknown.
- $obs(\tau)$ is the same sequence of states in τ but: (1), with certain states omitted and/or (2), the value of certain fluents omitted in that states, i.e. $|s_i| \leq |F|$ for every $0 \leq i \leq m$.

Constraint Satisfaction Problem

A *Constraint Satisfaction Problem* is defined as a 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 all variables. An evaluation is a *solution* for a given CSP $\langle X, D, C \rangle$ if it is consistent and complete.

Learning Temporal Planning Models

Our approach for learning temporal planning models from observations of plan executions is to compile this task intro a CSP task.

The learning task

First we formalize the task of learning temporal planning models from observations of plan executions with the pair $\Lambda = \langle \mathcal{M}, \omega \rangle$:

- $\mathcal{M}$ is the set of **initial action models**. This set is *empty*, when learning from scratch, or *partially specified*, when some fragments of the action models are known a priori.
- $\omega = \langle obs(\pi), obs(\tau) \rangle$ is a noiseless observation of a temporal plan execution such that:
  1. The initial state $s_0 \in obs(\tau)$ is a *fully observed* state including positive and negative fluents, i.e. $|s_0| = |F|$. Consequently, the corresponding set of predicates and objects that shape the fluents in F are inferrable from s_0.
  2. The header of an action model is either given by $\mathcal{M}$ or inferrable from $obs(\pi)$. In the latter case, $obs(\pi)$ must contain at least one instantiation of the respective action model header.

A *solution* to a learning task $\Lambda = \langle \mathcal{M}, \omega \rangle$ is a set of action models $\mathcal{M}'$ that is compliant with the input models $\mathcal{M}$ and the observed plan trace ω.

This definition of the learning task is extensible to the more general case where the execution of several plans from the same action models are observed. In this case, $\Lambda = \langle \mathcal{M}, \Omega \rangle$, where $\Omega = \{\omega_1, \ldots, \omega_k\}$ such that each $\omega \in \Omega$ is a plan trace that satisfies the previous assumptions. In this case, the *learned* action models $\mathcal{M}'$ have to be compliant with the input models $\mathcal{M}$ and also with every observed plan trace $\omega \in \Omega$.

Example

Figure 2 shows an example of a temporal plan for solving a problem from the *zenotravel domain*.

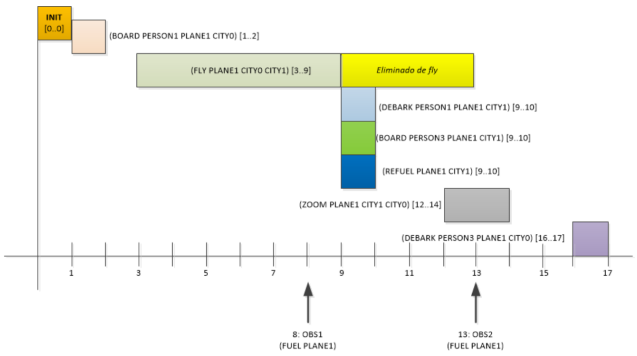

Figure 2: Example of a temporal plan for solving a problem from the *zenotravel domain*.

The compilation

Given an observation of the execution of temporal plans ω. 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 a singleton given by the input observation.
  - $duration_a$ representing the action duration. The domain of this variable is $[]$.
  - end_a, indicates the time-stamp for the end of the observed action a. The domain of this variable is $[]$.
  - $supports_{f,a_i}$ is a variable indicating that action a_i is the supporter of the fluent f in the observed plan. The domain of this variable is the set of actions.
  - $preS_{f,a}$ and $preE_{f,a}$ are Boolean variables indicating whether fuent f is precondition of the action a.
  - $time_{f,a}$ 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 $[]$.

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 $end_{a_n} = preS_{f,a_{a_n}})$. Constraints 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 set of action models $\mathcal{M}'$ that solves $\Lambda = \langle \mathcal{M}, \omega \rangle$ is computable in linear time and space.

Compilation properties

Lemma 1. *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 2. *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 a *fully* or *partially specified* temporal planning model $\mathcal{M}$ is given in Λ, the compilation also serves to validate whether the observed ω follows the given model $\mathcal{M}$:

- $\mathcal{M}$ is proved to be a *valid* action model for the given input data in ω 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 ω iff the output CSP is unsolvable. This means that $\mathcal{M}$ cannot be compliant with the given observation of the plan execution.

The 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.

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.

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.