Goal Recognition as Planning with Unknown Domain Models

Diego Aineto[1] , **Sergio Jiménez**[1] , **Eva Onaindia**[1] and , **Miquel Ramírez**[2]

[1]Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València. Valencia, Spain
[2]School of Computing and Information Systems. The University of Melbourne. Melbourne, Victoria. Australia

{dieaigar,serjice,onaindia}@dsic.upv.es, miquel.ramirez@unimelb.edu.au

Abstract

The paper shows how to relax one key assumption of the *plan recognition as planning* approach for *goal recognition* that is knowing the action model of the observed agents. The paper introduces a novel formulation for classical planning in a setting where no action model is given (instead, only the state variables and the action headers are given) and it shows how this formulation neatly fits with the *plan recognition as planning* approach. The empirical evaluation evidences that this novel formulation allows to solve standard goal recognition benchmarks without *a priori* knowing the action model of the observed agents and using an off-the-shelf classical planner.

1 Introduction

Goal recognition is a particular classification task in which each class represents a different goal and the classification examples are observations of agents acting to achieve one of that goals. Despite there exists a wide range of diffenrent approaches for *goal recognition*, *plan recognition as planning* [Ramírez and Geffner, 2009; Ramírez, 2012] is one of the most appealing since it is at the core of various activity recognition tasks such as, *goal recognition design* [Keren *et al.*, 2014], *deceptive planning* [Masters and Sardina, 2017], *planning for transparency* [MacNally *et al.*, 2018] or *counter-planning* [Pozanco *et al.*, 2018].

Plan recognition as planning leverages the action model of the observed agents and an off-the-shelf classical planner to compute the most likely goal of that agents. In this paper we show that we can relax the key assumption of the *plan recognition as planning* approach for *goal recognition* that is a priori having an action model of the observed agents. In particular, the paper introduces a novel formulation for classical planning in a setting where no action model is given (instead, only the state variables and the action headers are given) and it shows how this formulation neatly fits with the *plan recognition as planning* approach. The empirical evaluation evidences that this novel formulation allows to solve standard goal recognition benchmarks without *a priori* knowing the action model of the observed agents and using an off-the-shelf classical planner.

2 Background

This section formalizes the *planning model* we follow, the kind of *observations* that are given as input to the *goal recognition* task, and the *plan recognition as planning* approach for *goal recognition*.

2.1 Classical planning with conditional effects

Let F be the set of propositional state variables (*fluents*) 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). Given L, let $\neg L = \{\neg l : l \in L\}$ be its complement. 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 frame* is a tuple $\Phi = \langle F, A\rangle$, where F is a set of fluents and A is a set of *actions*. Each classical planning action $a \in A$ has a precondition $\mathsf{pre}(a) \in \mathcal{L}(F)$, a set of effects $\mathsf{eff}(a) \in \mathcal{L}(F)$, and a positive action cost $cost(a)$. The semantics of actions $a \in A$ is specified with two functions: $\rho(s, a)$ denotes whether action a is *applicable* in a state s and $\theta(s, a)$ denotes the *successor state* that results of applying action a in a state s. Then, $\rho(s, a)$ holds iff $\mathsf{pre}(a) \subseteq s$, i.e. if its precondition holds in s. The result of executing an applicable action $a \in A$ in a state s is a new state $\theta(s, a) = (s \setminus \neg\mathsf{eff}(a)) \cup \mathsf{eff}(a)$. Subtracting the complement of $\mathsf{eff}(a)$ from s ensures that $\theta(s, a)$ remains a well-defined state. The subset of action effects that assign a positive value to a state fluent is called *positive effects* and denoted by $\mathsf{eff}^+(a) \in \mathsf{eff}(a)$ while $\mathsf{eff}^-(a) \in \mathsf{eff}(a)$ denotes the *negative effects* of an action $a \in A$.

A *classical planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where I is the initial state and $G \in \mathcal{L}(F)$ is the set of goal conditions over the state variables. A *plan* π is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$, with $|\pi| = n$ denoting its *plan length* and $cost(\pi) = \sum_{a\in\pi} cost(a)$ its *plan cost*. The execution of π on the initial state I of P induces a *trajectory* $\tau(\pi, s_0) = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, it holds $\rho(s_{i-1}, a_i)$ and $s_i = \theta(s_{i-1}, a_i)$. A plan π solves P iff the induced *trajectory* $\tau(\pi, s_0)$ reaches a final state $G \subseteq s_n$, where all goal conditions are met. A solution plan is *optimal* iff its cost is minimal.

An *action with conditional effects* $a_c \in A$ is defined as a set of preconditions $\mathsf{pre}(a_c) \in \mathcal{L}(F)$ and a set of *conditional effects* $\mathsf{cond}(a_c)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a_c)$ is composed of two sets of literals: $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*. An action a_c is applicable in a state s if $\rho(s, a_c)$ is true, and the result of applying action a_c in state s is $\theta(s, a_c) = \{s \setminus \neg\mathsf{eff}_c(s,a) \cup \mathsf{eff}_c(s,a)\}$ where $\mathsf{eff}_c(s,a)$ are the *triggered effects* resulting from the action application (conditional effects whose conditions hold in s):

$$\mathsf{eff}_c(s,a) = \bigcup_{C \rhd E \in \mathsf{cond}(a_c), C \subseteq s} E,$$

2.2 The observation model

Given a planning problem $P = \langle F, A, I, G \rangle$, a plan π and a trajectory $\tau(\pi, P)$, we define the *observation of the trajectory* as an interleaved combination of actions and states that represents the observation from the execution of π in P. Formally, $\mathcal{O}(\tau) = \langle s_0^o, a_1^o, s_1^o \ldots, a_l^o, s_m^o \rangle$, $s_0^o = I$, and:

- The **observed actions** are consistent with π, which means that $\langle a_1^o, \ldots, a_l^o \rangle$ is a sub-sequence of π. Specifically, the number of observed actions, l, can range from 0 (fully unobservable action sequence) to $|\pi|$ (fully observable action sequence).
- The **observed states** $\langle s_0^o, s_1^o, \ldots, s_m^o \rangle$ is a sequence of possibly *partially observable states*, except for the initial state s_0^o, which is fully observable. A partially observable state s_i^o is one in which $|s_i^o| < |F|$; i.e., a state in which at least a fluent of F is not observable. Note that this definition also comprises the case $|s_i^o| = 0$, when the state is fully unobservable. Whatever the sequence of observed states of $\mathcal{O}(\tau)$ is, it must be consistent with the sequence of states of $\tau(\pi, P)$, meaning that $\forall i, s_i^o \subseteq s_i$. In practice, the number of observed states, m, range from 1 (the initial state, at least), to $|\pi|+1$, and the observed intermediate states will comprise a number of fluents between $[1, |F|]$.

We assume a bijective monotone mapping between actions/states of trajectories and observations [Ramírez and Geffner, 2009], thus also granting the inverse consistency relationship (the trajectory is a superset of the observation). Therefore, transiting between two consecutive observed states in $\mathcal{O}(\tau)$ may require the execution of more than a single action ($\theta(s_i^o, \langle a_1, \ldots, a_k \rangle) = s_{i+1}^o$, where $k \geq 1$ is unknown but finite. In other words, having $\mathcal{O}(\tau)$ does not imply knowing the actual length of π.

2.3 Goal recognition with classical planning

Goal recognition is a particular classification task in which each class represents a different goal $g \in G[\cdot]$ and there is a single classification example $\mathcal{O}(\tau)$ that represents the observation of an agent acting to achieve one of the input goals in $g \in G[\cdot]$. Following the *naive Bayes classifier*, the *solution* to the *goal recognition* task is the subset of goals in $G[\cdot]$ that maximizes this expression.

$$argmax_{g \in G[\cdot]} P(\mathcal{O}|g)P(g). \quad (1)$$

The *plan recognition as planning* approach shows how to compute estimates of the $P(\mathcal{O}|g)$ likelihood leveraging the action model of the observed agent and an off-the-shelf classical planner. More precisely, given a *classical planning problem* $P = \langle F, A, I, G[\cdot] \rangle$, where $G[\cdot]$ represents the set of possible goals, then the *plan recognition as planning* approach estimates the $P(\mathcal{O}|g)$. This estimate is computed by calculating, for each goal $g \in G[\cdot]$, the cost difference of the solutions to these two different classical planning problems:

- $P_g^\top$, that is a classical planning problem built constraining $P = \langle F, A, I, g \rangle$ to achieve the particular goal $g \in G[\cdot]$ through a plan $\pi^\top$ *consistent* with the input observation $\mathcal{O}(\tau)$.
- $P_g^\perp$, that constrains $P = \langle F, A, I, g \rangle$ to achieve $g \in G[\cdot]$ through a plan $\pi^\perp$ *inconsistent* with $\mathcal{O}(\tau)$.

The higher the value of this cost difference $cost(\pi^\top) - cost(\pi^\perp)$, the higher probability of aiming to achieve the $g \in G[\cdot]$ goal. With this regard, *plan recognition as planning* uses the *sigmoid function* to map the previous cost difference into a likelihood:

$$P(\mathcal{O}|g) = \frac{1}{1 + e^{-\beta(cost(\pi^\top) - cost(\pi^\perp))}} \quad (2)$$

This expression is derived from the assumption that while the observed agent is not perfectly rational, he is more likely to follow cheaper plans, according to a *Boltzmann* distribution. The larger the value of β, the more rational the agent, and the less likely that he will follow suboptimal plans. Recent works show that estimates of the $P(\mathcal{O}|g)$ likelihood can be faster computed using relaxations of the classical planning tasks [Pereira *et al.*, 2017].

The original work on *plan recognition as planning* assumes a less expressive observation model where observations are referred only about logs of executed actions [Ramírez and Geffner, 2009]. However the same approach applies to more expressive observation models that consider also state observations, like the one defined above, with a trivial three-fold extension:

- One fluent $\{validated_j\}_{0 \leq j \leq m}$ to point at every $s_j \in \mathcal{O}(\tau)$ state observation.
- Adding $validated_m$ to every possible goal $g \in G[\cdot]$ to constraint the solution plans $\pi^\top$ to be consistent with all the state observations.
- One $\mathsf{validate_j}$ action to constraint $\pi^\top$ to be consistent with the $s_j \in \mathcal{O}(\tau)$ input state observation, $(1 \leq j \leq m)$.

$$\mathsf{pre}(\mathsf{validate_j}) = s_j \cup \{validated_{j-1}\},$$
$$\mathsf{cond}(\mathsf{validate_j}) = \{\emptyset\} \rhd \{\neg validated_{j-1}, validated_j\}.$$

3 Classical planning with unknown domain models

This section defines a classical planning problem $P = \langle F, A[\cdot], I, G \rangle$, where $A[\cdot]$ is a set of actions s.t., the semantics of each action $a \in A[\cdot]$ is unknown (i.e. the functions

ρ and/or θ of a are undefined but the corresponding action headers are known).

A solution to this task is a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ whose execution induces a *trajectory* $\tau(\pi, I) = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$ such that $s_0 = I$ and it holds that *there exists* at least one model in the space of possible action models (e.g. one possible definition of the ρ and θ functions with the given state variables and action headers) that satisfies that $\rho(s_{i-1}, a_i)$ and $s_i = \theta(s_{i-1}, a_i)$, for each $1 \le i \le n$, as well as it reaches a final state where all goal conditions are met, $G \subseteq s_n$.

First, se show how to encode the space of possible models for the actions in $A[\cdot]$ with a set of propositional variables and a set of constraints over that variables and finally, we show how to exploit this encoding to compute a solution to the $P = \langle F, A[\cdot], I, G \rangle$ problem with an off-the-shelf classical planner.

3.1 A propositional encoding for the space of STRIPS action models

A STRIPS *action schema* ξ is defined by four lists: A list of *parameters* $pars(\xi)$, and three list of predicates (namely $pre(\xi)$, $del(\xi)$ and $add(\xi)$) that shape the kind of fluents that can appear in the *preconditions*, *negative effects* and *positive effects* of the actions induced from that schema. Let be Ψ the set of *predicates* that shape the propositional state variables F, and a list of *parameters* $pars(\xi)$. The set of elements that can appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of the STRIPS action schema ξ is given by FOL interpretations of Ψ over the parameters $pars(\xi)$ and is denoted as $\mathcal{I}_{\Psi,\xi}$.

For instance, in the *blocksworld* the $\mathcal{I}_{\Psi,\xi}$ set contains only five elements for a `pickup(`v_1`)` schemata, $\mathcal{I}_{\Psi,pickup}$={`handempty`, `holding(`$v_1$`)`, `clear(`$v_1$`)`, `ontable(`$v_1$`)`, `on(`$v_1$`,`$v_1$`)`} while it contains eleven elements for a `stack(`v_1`,`v_2`)` schemata, $\mathcal{I}_{\Psi,stack}$={`handempty`, `holding(`v_1`)`, `holding(`v_2`)`, `clear(`v_1`)`, `clear(`v_2`)`, `ontable(`v_1`)`, `ontable(`v_2`)`, `on(`v_1`,`v_1`)`, `on(`v_1`,`v_2`)`, `on(`v_2`,`v_1`)`, `on(`v_2`,`v_2`)`}.

Despite any element of $\mathcal{I}_{\Psi,\xi}$ can *a priori* appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of schema ξ, the space of possible STRIPS schemata is bounded by constraints of three kinds:

1. *Syntactic constraints*. STRIPS constraints require $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$. Considering exclusively these syntactic constraints, the size of the space of possible STRIPS schemata is given by $2^{2\times|\mathcal{I}_{\Psi,\xi}|}$. *Typing constraints* are also of this kind [McDermott *et al.*, 1998].
2. *Domain-specific constraints*. One can introduce domain-specific knowledge to constrain further the space of possible schemata. For instance, in the *blocksworld* one can argue that `on(`v_1`,`v_1`)` and `on(`v_2`,`v_2`)` will not appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ lists of an action schema ξ because, in this specific domain, a block cannot be on top of itself. *State invariants* are also constraints of this kind [Fox and Long, 1998].

```
(:action stack
   :parameters (?v1 ?v2)
   :precondition (and (holding ?v1) (clear ?v2))
   :effect (and (not (holding ?v1)) (not (clear ?v2))
                (clear ?v1) (handempty) (on ?v1 ?v2)))


(pre_holding_v1_stack) (pre_clear_v2_stack)
(eff_holding_v1_stack) (eff_clear_v2_stack)
(eff_clear_v1_stack) (eff_handempty_stack) (eff_on_v1_v2_stack)
```

Figure 1: PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema.

3. *Observation constraints*. An observations $\mathcal{O}(\tau)$ depicts *semantic knowledge* that constraints further the space of possible action schemata.

In this work we introduce a propositional encoding of the *preconditions*, *negative*, and *positive* effects of a STRIPS action schema ξ using only fluents of two kinds `pre_e_`ξ and `eff_e_`ξ (where $e \in \mathcal{I}_{\Psi,\xi}$). This encoding exploits the syntactic constraints of STRIPS so is more compact that the one previously proposed by Aineto *et al.* 2018. In more detail, if `pre_e_`ξ and `eff_e_`ξ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a negative effect in ξ while if pre_e_ξ does not hold but `eff_e_`ξ holds, it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a positive effect in ξ. Figure 1 shows the PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema with `pre_e_stack` and `eff_e_stack` fluents ($e \in \mathcal{I}_{\Psi,stack}$).

3.2 A classical planning compilation for planning with unknown domain models

To solve a classical planning problem $P = \langle F, A[\cdot], I, G \rangle$ we create another classical planning problem $P = \langle F', A', I, G \rangle$ such that:

- F' extends F with the necessary fluents for the propositional encoding of the corresponding space of STRIPS action models. This is a set of fluents of the type $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ such that $e \in \mathcal{I}_{\Psi,\xi}$ is a single element from the set of FOL interpretations of predicates Ψ over the corresponding parameters $pars(\xi)$.
- A' replaces the actions in A with two new types of actions.
  1. Actions for *inserting* a component (precondition, positive effect or negative effect) in $\xi \in \mathcal{M}$ following the syntactic constraints of STRIPS models.
     - Actions which support the addition of a *precondition* $p \in \Psi_\xi$ to the action model $\xi \in \mathcal{M}$. A precondition p is inserted in ξ when neither pre_p, eff_p exist in ξ.

$$\begin{aligned}\mathsf{pre}(\mathsf{insertPre}_{\mathsf{p},\xi}) =& \{\neg pre_p(\xi), \neg eff_p(\xi)\},\\ \mathsf{cond}(\mathsf{insertPre}_{\mathsf{p},\xi}) =& \{\emptyset\} \rhd \{pre_p(\xi)\}.\end{aligned}$$

     - Actions which support the addition of a *negative* or *positive* effect $p \in \Psi_\xi$ to the action model $\xi \in \mathcal{M}$.

$$\begin{aligned}\mathsf{pre}(\mathsf{insertEff}_{\mathsf{p},\xi}) =& \{\neg eff_p(\xi)\},\\ \mathsf{cond}(\mathsf{insertEff}_{\mathsf{p},\xi}) =& \{\emptyset\} \rhd \{eff_p(\xi)\}.\end{aligned}$$

2. Actions for *applying* the action models $\xi \in \mathcal{M}$ built by the insert actions and bounded to objects $\omega \subseteq \Omega^{ar(\xi)}$. Since action headers are known, the variables $pars(\xi)$ are bounded to the objects in ω that appear in the same position.

$$\begin{aligned}\mathsf{pre}(\mathsf{apply}_{\xi,\omega}) =& \{pre_p(\xi) \implies p(\omega)\}_{\forall p\in\Psi_\xi},\\ \mathsf{cond}(\mathsf{apply}_{\xi,\omega}) =& \{pre_p(\xi) \wedge eff_p(\xi)\} \rhd \{\neg p(\omega)\}_{\forall p\in\Psi_\xi},\\ & \{\neg pre_p(\xi) \wedge eff_p(\xi)\} \rhd \{p(\omega)\}_{\forall p\in\Psi_\xi}.\end{aligned}$$

```
(:action apply_stack
  :parameters (?o1 - object ?o2 - object)
  :precondition
   (and (or (not (pre_stack_on_v1_v1)) (on ?o1 ?o1))
        (or (not (pre_stack_on_v1_v2)) (on ?o1 ?o2))
        (or (not (pre_stack_on_v2_v1)) (on ?o2 ?o1))
        (or (not (pre_stack_on_v2_v2)) (on ?o2 ?o2))
        (or (not (pre_stack_ontable_v1)) (ontable ?o1))
        (or (not (pre_stack_ontable_v2)) (ontable ?o2))
        (or (not (pre_stack_clear_v1)) (clear ?o1))
        (or (not (pre_stack_clear_v2)) (clear ?o2))
        (or (not (pre_stack_holding_v1)) (holding ?o1))
        (or (not (pre_stack_holding_v2)) (holding ?o2))
        (or (not (pre_stack_handempty)) (handempty)))
  :effect
   (and (when (del_stack_on_v1_v1) (not (on ?o1 ?o1)))
        (when (del_stack_on_v1_v2) (not (on ?o1 ?o2)))
        (when (del_stack_on_v2_v1) (not (on ?o2 ?o1)))
        (when (del_stack_on_v2_v2) (not (on ?o2 ?o2)))
        (when (del_stack_ontable_v1) (not (ontable ?o1)))
        (when (del_stack_ontable_v2) (not (ontable ?o2)))
        (when (del_stack_clear_v1) (not (clear ?o1)))
        (when (del_stack_clear_v2) (not (clear ?o2)))
        (when (del_stack_holding_v1) (not (holding ?o1)))
        (when (del_stack_holding_v2) (not (holding ?o2)))
        (when (del_stack_handempty) (not (handempty)))
        (when (add_stack_on_v1_v1) (on ?o1 ?o1))
        (when (add_stack_on_v1_v2) (on ?o1 ?o2))
        (when (add_stack_on_v2_v1) (on ?o2 ?o1))
        (when (add_stack_on_v2_v2) (on ?o2 ?o2))
        (when (add_stack_ontable_v1) (ontable ?o1))
        (when (add_stack_ontable_v2) (ontable ?o2))
        (when (add_stack_clear_v1) (clear ?o1))
        (when (add_stack_clear_v2) (clear ?o2))
        (when (add_stack_holding_v1) (holding ?o1))
        (when (add_stack_holding_v2) (holding ?o2))
        (when (add_stack_handempty) (handempty))
        (when (modeProg) (not (modeProg)))))
```

Figure 2: PDDL action for applying an already programmed model for *stack* (implications are coded as disjunctions).

The dynamics of the actions for *applying* an action model $\xi \in \mathcal{M}$ is determined by the values of the model the $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e\in\mathcal{I}_{\Psi,\xi}}$ fluents in the current state. Figure 2 shows the PDDL encoding of `(apply_stack)` for applying the action model of the *stack* operator.

For instance, executing the action `(apply_stack blockB blockA)` in a state s implies activating the preconditions and effects of `(apply_stack)` according to the values of the model fluents in s. This means that if the current state s holds `{(pre_stack_holding_v1),(pre_stack_clear_v2)}` $\subset s$, then it must be checked that positive literals `(holding blockB)` and `(clear blockA)` hold in s. Otherwise, a different set of precondition literals will be checked. The same applies to the conditional effects, generating the corresponding literals according to the values of the model fluents of s. Note that executing `(apply_stack blockB blockA)`, will add the literals `(on blockB blockA)`, `(clear blockB)`, `(not(clear blockA))`, `(handempty)` and `(not(clear blockB))` to the successor state if `stack` has been correctly programmed by the insert actions.

3.3 Compilation properties

Now we present some theoretical properties of the compilation scheme.

Soundness and completeness

Lemma 1 *Soundness. Any classical plan π_Λ that solves P_Λ induces a set of action models $\mathcal{M}'$ that solves $\Lambda = \langle\mathcal{M}, \tau\rangle$.*

[Proof sketch] Once action models $\mathcal{M}'$ are programmed, they can only be applied and validated because of the $mode_{prog}$ fluent. In addition, P_Λ is only solvable if fluents at_n and $test_m$ hold at the last state reached by π_Λ. By the definition of the $\mathsf{apply}_{\xi,\omega}$ and the $\mathsf{validate_j}$ actions, these goals can only be achieved executing an applicable sequence of programmed action models that reaches every state $s_j \in \tau$, starting in the corresponding initial state and following the sequence of n observed actions of τ. This means that the programmed action model $\mathcal{M}'$ is consistent with the provided input knowledge and hence, that $\mathcal{M}'$ is a solution to Λ.

Lemma 2 *Completeness. Any set of action models $\mathcal{M}'$ that solves $\Lambda = \langle\mathcal{M}, \tau\rangle$ is computable solving the corresponding classical planning task P_Λ.*

[Proof sketch] By definition, $\Psi_\xi \subseteq \Psi_v$ fully captures the set of elements that can appear in an action model $\xi \in \mathcal{M}$. The compilation does not discard any possible set of action models $\mathcal{M}'$ definable within Ψ_v that satisfies the observed state trajectory and action sequence of τ. This means that for every $\mathcal{M}'$ that solves Λ, there exists a plan π_Λ that can be built selecting the appropriate programming, apply and validate actions from the P_Λ compilation.

Size

The size of the planning task P_Λ output by the compilation approach depends on:

- The arity of the actions and the fluents in τ given as input in Λ. The larger the arity, the larger the size of the Ψ_ξ sets. This is the term that dominates the compilation size because it defines the $pre_p(\xi)/del_p(\xi)/add_p(\xi)$ fluents and the corresponding set of *programming* actions.
- The length of the observed action sequence and state trajectory of τ. The larger the number of observed actions, $a_i \in \tau$ s.t. $1 \leq i \leq n$, the more $\{at_i\}$ fluents. The larger the number of observed states, $s_j \in \tau$ s.t. $1 \leq j \leq m$, the more $\{test_j\}$ fluents and $\{\mathsf{validate_j}\}$ actions in P_Λ.

The bias of the initially empty action model

Since in the initial state of the classical planning compilation all the $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e\in\mathcal{I}_{\Psi,\xi}}$ are false, our compilation introduces a bias to solve the $P = \langle F, A[\cdot], I, G\rangle$ classical planning task. This bias can be eliminated definin a cost landscape where any actions for determinign a precondition or an action effect has zero cost. Since classical planners are nor proficincy when optimizing the cost of solutions with this kind of cost landscapes we use a different aprpoach to disregard the cost of the actions that add a precondtion/effect to an action model. Our approach is to use a SAT-based planner that applies the actions for programming preconditions in a single planning step (in parallel) because these actions do not interact. Actions for programming action effects can also be applied in a single planning step so the plan horizon for programming any action model is always 2 which in adddition, significantly reduces the planning horizon.

4 Goal recognition as planning with unknown domain models

We define the task of *goal recognition with unknown domain models* as a $\langle P, \mathcal{O}(\tau)\rangle$ pair, where:

- $P = \langle F, A[\cdot], I, G[\cdot]\rangle$ is a planning problem where $G[\cdot]$ is the set of possible goals and $A[\cdot]$ is a set of actions s.t., for each $a \in A[\cdot]$, the semantics of a is unknown (i.e. the functions ρ and/or θ of a are undefined).
- $\mathcal{O}(\tau)$ is an observation of a trajectory $\tau(\pi, I)$ produced by the execution of an unknown plan π that reaches a goal $g \in G[\cdot]$ starting from the given initial state.

The *solution* to the *goal recognition with unknown domain models* task is again the subset of goals in $G[\cdot]$ that maximizes expression (1).

4.1 Computing the $P(\mathcal{O}|g)$ with unknown domain models

Now we are ready to compute the target distribution $P(g|\mathcal{O})$ over the possible goals $g \in G[\cdot]$ given the observation $\mathcal{O}(\tau)$:

1. For each goal, we define the $P^{\top}$, that constrains the classical planning problem $P = \langle F, A[\cdot], s_0, g\rangle$ to achieve $g \in G[\cdot]$ through a plan $\pi^{\top}$ *consistent* with the input observation $\mathcal{O}(\tau)$. Note that s_0 is the initial state in the given observation $\mathcal{O}(\tau)$. We use our adapted compilation to compute the classcial planning tasks $P_{\lambda}^{\top}$ and solve them using an off-the-shelf-classical planner.
2. For each goal, we define $P^{\perp}$, that constrains $P = \langle F, A, s_0, g\rangle$ to achieve $g \in G[\cdot]$ through a plan $\pi^{\perp}$ *inconsistent* with $\mathcal{O}(\tau)$ and that uses the action model A used by the corresponding solution $\pi^{\top}$.
3. We compute the cost difference $\Delta(cost(\pi_{\top}), cost(\pi_{\perp}))$ where these costs are defined as the length of the postfix of the $\pi_{\lambda}^{\top}$ and $\pi_{\lambda}^{\perp}$ plans and plug this cost difference into equation (2) to get the $P(g|\mathcal{O})$ likelihoods.
4. Finally the previous likelihoods are plugged into the Bayes rule from which the goal posterior probabilities are obtained. In this case the $P\mathcal{O}(\tau)$ probabilities are obtained by normalization (goal probabilities must add up to 1 when summed over all possible goals).

5 Evaluation

6 Conclusions

References

[Aineto *et al.*, 2018] Diego Aineto, Sergio Jiménez, and Eva Onaindia. Learning STRIPS action models with classical planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, pages 399–407. AAAI Press, 2018.

[Fox and Long, 1998] Maria Fox and Derek Long. The automatic inference of state invariants in tim. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.

[Keren *et al.*, 2014] Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design. In *International Conference on Automated Planning and Scheduling, (ICAPS-14)*, pages 154–162, 2014.

[MacNally *et al.*, 2018] Aleck M MacNally, Nir Lipovetzky, Miquel Ramirez, and Adrian R Pearce. Action selection for transparent planning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1327–1335. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[Masters and Sardina, 2017] Peta Masters and Sebastian Sardina. Deceptive path-planning. In *IJCAI 2017*, pages 4368–4375. AAAI Press, 2017.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language, 1998.

[Pereira *et al.*, 2017] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. Landmark-based heuristics for goal recognition. In *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*. AAAI Press, 2017.

[Pozanco *et al.*, 2018] Alberto Pozanco, Yolanda E.-Martín, Susana Fernández, and Daniel Borrajo. Counterplanning using goal recognition and landmarks. In *International Joint Conference on Artificial Intelligence, (IJCAI-18)*, pages 4808–4814, 2018.

[Ramírez and Geffner, 2009] Miquel Ramírez and Hector Geffner. Plan recognition as planning. In *International Joint conference on Artifical Intelligence, (IJCAI-09)*, pages 1778–1783. AAAI Press, 2009.

[Ramírez, 2012] Miquel Ramírez. *Plan recognition as planning*. PhD thesis, Universitat Pompeu Fabra, 2012.