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

This paper shows how to relax a strong assumption of the *plan recognition as planning* approach 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 parameters are known) and it shows that this formulation neatly fits with the *plan recognition as planning* approach for *goal recognition*. The empirical evaluation evidences that this novel formulation allows to solve standard goal recognition benchmarks, still using an off-the-shelf classical planner, but without *a priori* knowing the action model of the observed agents.

1 Introduction

Goal recognition is a particular classification task in which each class represents a different goal and classification examples are observations of agents acting to achieve one of those goals. Despite there exists a wide range of different approaches for *goal recognition*, *plan recognition as planning* [Ramírez and Geffner, 2009; Ramírez, 2012] is one of the most popular and it is currently at the core of various model-based 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 how to relax the assumption of *knowing the action model of the observed agents*, which frequently becomes a too strong assumption when applying *plan recognition as planning* at real-world problems. 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 parameters are known) and it shows that this formulation neatly fits with the successful *plan recognition as planning* approach. The empirical evaluation evidences that this novel formulation allows to solve standard goal recognition benchmarks, still using an off-the-shelf classical planner, but without *a priori* knowing the action model of the observed agents.

2 Background

This section formalizes the *planning model* we follow, the kind of *observations* that input 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 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 of P induces a *trajectory* $\tau(\pi, P) = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ 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* $\tau(\pi, P)$ reaches a final state $G \subseteq s_n$, where all goal conditions are met. A solution plan is *optimal* iff its cost is minimal.

We also define *actions with conditional effects* because they are useful to compactly formulate our approach for *goal recognition with unknown domain models*. An action $a_c \in A$ with conditional effects is 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, $\mathcal{O}(\tau) = \langle s^o_0, a^o_1, s^o_1 \ldots, a^o_l, s^o_m\rangle$, $s^o_0 = I$, and:

- The **observed actions** are consistent with π, which means that $\langle a^o_1, \ldots, a^o_l\rangle$ is a sub-sequence of π. The number of observed actions, l, ranges from 0 (fully unobserved action sequence) to $|\pi|$ (fully observed action sequence).
- The **observed states** $\langle s^o_0, s^o_1, \ldots, s^o_m\rangle$ is a sequence of possibly *partially observable states*, except for the initial state s^o_0, which is fully observed. A partially observable state s^o_i is one in which $|s^o_i| < |F|$; i.e., a state in which at least a fluent of F is not observable. Note that this definition also comprises the case $|s^o_i| = 0$, 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 $\forall i, s^o_i \subseteq s_i$. The number of observed states, m, range from 1 (the initial state, at least), to $|\pi| + 1$, and each *observed* states comprises $[1, |F|]$ fluents (the observation can still miss intermediate states that are *unobserved*).

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 ($\theta(s^o_i, \langle a_1, \ldots, a_k\rangle) = s^o_{i+1}$, where $k \geq 1$ is unknown but finite. In other words, having an input observation $\mathcal{O}(\tau)$ 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 possible goal $G \in G[\cdot]$ and there is a single classification example, $\mathcal{O}(\tau)$, that represents the observation of agents acting to achieve a goal $G \in G[\cdot]$.

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 that the $P(\mathcal{O}|G)$ likelihood can be estimated leveraging the action model of the observed agents and an off-the-shelf classical planner [Ramírez, 2012]. Given a *classical planning problem* $P = \langle F, A, I, G[\cdot]\rangle$ (where $G[\cdot]$ represents the set of *recognizable* goals) then $P(\mathcal{O}|G)$ is estimated computing, for each goal $G \in G[\cdot]$, the cost difference of the solution plans to these two classical planning problems:

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

The higher the value of the $cost(\pi^{\top}) - cost(\pi^{\perp})$ difference, the higher the probability of the observed agents to aim goal $G \in G[\cdot]$. 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 agents are not perfectly rational, they are more likely to follow cheaper plans, according to a *Boltzmann* distribution. The larger the value of β, the more rational the agents, and the less likely that they will follow suboptimal plans. Recent works show fast estimates of the $P(\mathcal{O}|G)$ likelihood computed using relaxations of the classical planning tasks [Pereira *et al.*, 2017].

3 Planning with unknown domain models

This section introduces a novel formulation for classical planning in a setting where no action model is given. This setting has already shown related to the learning of action models for planning [Stern and Juba, 2017]. In particular it can be seen as an extreme scenario when the action model is learned from a single example that contains only two state observations: the initial state and the goals. A *classical planning with unknown domain models* is then a tuple $P = \langle F, A[\cdot], I, G\rangle$, 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).

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 *there exists* at least one possible action model (e.g. one possible definition of the ρ and θ functions within the given state variables) that satisfies $\rho(s_{i-1}, a_i)$ and $s_i = \theta(s_{i-1}, a_i)$, for every $1 \leq i \leq n$, and such that the reached final state meets the goal conditions, $G \subseteq s_n$.

Next we show that the space of possible STRIPS action models can be encoded as a set of propositional variables and

a set of constraints over those variables. Then, we show how to exploit this encoding to solve $P = \langle F, A[\cdot], I, G \rangle$ problems with an off-the-shelf classical planner and the properties of this approach.

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 a four-operator *blocksworld* [Slaney and Thiébaux, 2001], the $\mathcal{I}_{\Psi,\xi}$ set contains only five elements for the `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 the `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 actual 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].
3. **Observation constraints**. An observation $\mathcal{O}(\tau)$ 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 it is more compact that the one previously proposed by Aineto *et al.* 2018 for learning classical planning action models. In more detail, if `pre_e_`ξ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *precondition* in ξ. 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 ($e \in \mathcal{I}_{\Psi,stack}$).

```
(: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.

3.2 A classical planning compilation for planning with unknown domain models

Now we show how we adapt the *classical planning compilation for learning* STRIPS *action models* [Aineto *et al.*, 2018] to address the task of *planning with unknown domain models*, using our propositional encoding of STRIPS action models.

Given a classical planning problem with unknown domain models $P = \langle F, A[\cdot], I, G \rangle$ we create a classical planning problem $P' = \langle F', A', I, G \rangle$ such that:

- F' extends F with a fluent $mode_{insert}$, to indicate whether action models are being programmed, and the 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 action parameters $pars(\xi)$.
- A' replaces the actions in A with two types of actions.
  1. Actions for *inserting* a *precondition*, *positive* effect or *negative* effect in ξ following the syntactic constraints of STRIPS models.
     - Actions which support the addition of a *precondition* $p \in \Psi_\xi$ to the action model ξ. A precondition p is inserted in ξ when neither pre_p, eff_p exist in ξ.

$$\mathsf{pre}(\mathsf{insertPre_{p,\xi}}) = \{\neg pre_p(\xi), \neg eff_p(\xi), mode_{insert}\},$$
$$\mathsf{cond}(\mathsf{insertPre_{p,\xi}}) = \{\emptyset\} \rhd \{pre_p(\xi)\}.$$

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

$$\mathsf{pre}(\mathsf{insertEff_{p,\xi}}) = \{\neg eff_p(\xi), mode_{insert}\},$$
$$\mathsf{cond}(\mathsf{insertEff_{p,\xi}}) = \{\emptyset\} \rhd \{eff_p(\xi)\}.$$

  2. Actions for *applying* an action model ξ built by the *insert* actions and bounded to objects $\omega \subseteq \Omega^{|pars(\xi)|}$ (where Ω is the set of *objects* used to induce the fluents F by assigning objects in Ω to the Ψ predicates and Ω^k is the k-th Cartesian power of

Ω). The action parameters, $pars(\xi)$, are bound to the objects in ω that appear in the same position.

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

The intuition of the compilation is that the dynamics of the actions for *applying* an action model ξ is determined by the values of the corresponding $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ fluents in the current state. For instance, executing `(apply_stack blockB blockA)` in a state s implies activating the preconditions and effects of `apply_stack` according to the values of $\{pre\_e\_stack, eff\_e\_stack\}_{\forall e \in \mathcal{I}_{\Psi,stack}}$ fluents in s. This means that if the current state s holds $\{(\texttt{pre\_stack\_holding\_v1}), (\texttt{pre\_stack\_clear\_v2})\} \subset s$, 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 for the stack action. The same applies to the positive and negative effects. 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 only if `stack` has been correctly programmed by the *insert* actions.

```
00: (insert_pre_stack_holding_v1)
01: (insert_pre_stack_clear_v2)
02: (insert_eff_stack_clear_v1)
03: (insert_eff_stack_clear_v2)
04: (insert_eff_stack_handempty)
05: (insert_eff_stack_holding_v1)
06: (insert_eff_stack_on_v1_v2)
07: (insert_pre_pickup_clear_v1)
08: (insert_pre_pickup_ontable_v1)
09: (insert_pre_pickup_handempty)
10: (insert_eff_pickup_clear_v1)
11: (insert_eff_pickup_ontable_v1)
12: (insert_eff_pickup_handempty)
13: (insert_eff_pickup_holding_v1)
14: (apply_pickup blockB)
15: (apply_stack blockB blockC)
16: (apply_pickup blockA)
17: (apply_stack blockA blockB)
```

Figure 2: Plan computed when solving the classical planning problem output by our compilation corresponding to a classical planning with unknown domain models.

Figure 2 shows a solution plan computed when solving a $P' = \langle F', A', I, G\rangle$ classical planning problem output by our compilation. In the initial state of that problem three blocks (namely `blockA`,`blockB` and `blockC`) are clear and on top of the table and the robot hand is empty. The problem goals is having the 3-block tower `blockA` on top of `blockB` and `blockB` on top of `blockC`. The plan shows that the *insert* actions for the action model `stack` (steps $00-01$ insert the preconditions of the `stack` model, steps $02-06$ insert the action model effects), steps $07-13$ insert the preconditions and effects of the `pickup` action model and finally, steps $14-17$ is the plan postfix that applies the programmed action model to achieve the goals G. Note that another valid solution could be computed for instance, inserting the same preconditions and effects into the *putdown* and *unstack* action models and then applying instead the four step postfix `(putdown blockB)`, `(unstack blockB blockC)`, `(putdown blockA)`, `(unstack blockA blockB)`.

3.3 Compilation properties

Now we present some properties of the compilation scheme.

Lemma 1. *Soundness. Any classical plan π' that solves P' produces a solution to the classical planning problem with unknown domain models $P = \langle F, A[\cdot], I, G\rangle$.*

Proof. Once a given precondition or effect is inserted into an action model it can never be removed back and once an action model is applied it cannot be *reprogramed*. In the compiled problem the value of F, the fluents of the original problem, can exclusively be modified via $\mathsf{apply}_{\xi,\omega}$ actions. The set of goals G can only be achieved executing an applicable sequence of $\mathsf{apply}_{\xi,\omega}$ actions that, starting in the corresponding initial state reach a state $G \subseteq s_n$. This means that the action model used by the $\mathsf{apply}_{\xi,\omega}$ actions has to be consistent with the traversed intermediate states. We know that this must be true by the definition of the $\mathsf{apply}_{\xi,\omega}$ so hence, the sub-sequence of $\mathsf{apply}_{\xi,\omega}$ appearing in π' to solve P' is a solution plan to $P = \langle F, A[\cdot], I, G\rangle$. □

Lemma 2. *Completeness. Any plan π that solves $P = \langle F, A[\cdot], I, G\rangle$ is computable solving the corresponding classical planning task P'.*

Proof. By definition $\mathcal{I}_{\Psi,\xi}$ fully captures the set of elements that can appear in an action model ξ using predicates Ψ. Furthermore, the compilation does not discard any possible action model definable within $\mathcal{I}_{\Psi,\xi}$. This means that for every plan π that solves $P = \langle F, A[\cdot], I, G\rangle$, we can build a plan π' by selecting the appropriate actions for inserting precondition and effects to the corresponding action model and then selecting the corresponding $\mathsf{apply}_{\xi,\omega}$ actions that transform the initial state I into a state $G \subseteq s_n$. □

The bias of an initially *empty* action model

Any $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ fluent is false at the initial state of our compilation. This fact can introduce a bias to the solutions of the $P = \langle F, A[\cdot], I, G\rangle$ classical planning task preferring solutions that imply action models with a smaller number of *preconditions/effects* (i.e., that imply a lower number of *insert* actions).

This bias might be eliminated defining a cost landscape where *insert* actions has *zero cost* while $\mathsf{apply}_{\xi,\omega}$ actions has a *positive constant cost*. In practice, since classical planners are not proficiency optimizing cost landscapes of this kind, we use a different approach that disregard the cost of actions the *insert* actions. Our approach is to use SAT-based planning because it can apply all the required actions for inserting preconditions in a single planning step (in parallel), because these actions do not interact. Further, actions for inserting action effects are also applied in a single planning step so the plan horizon for programming any action model is always bound to 2, which significantly reduces the planning horizon. The SAT-based planning approach is also convenient because its ability to deal with populated with dead-ends and because symmetries in the insertion of preconditions/effects into an action model do not affect to performance.

Compilation size

The size of the classical planning task P' output by our compilation depends on the arity of the given *predicates* Ψ, that shape the propositional state variables F, and the number of parameters of the action models, $|pars(\xi)|$. The larger these arities, the larger $|\mathcal{I}_{\Psi,\xi}|$. The size of the $\mathcal{I}_{\Psi,\xi}$ set is the term that dominates the compilation size because it defines the $pre_p(\xi)/del_p(\xi)/add_p(\xi)$ fluents, the corresponding set of *insert* actions, and the number of conditional effects in the $\mathsf{apply}_{\xi,\omega}$ actions.

Note that *typing* can be used straightforward to constrain the FOL interpretations of Ψ over the parameters $pars(\xi)$ which significantly reduces $|\mathcal{I}_{\Psi,\xi}|$ and hence, the size of the classical planning task output by the compilation.

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 classical planning problem where $G[\cdot]$ is the set of *recognizable* 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 $\tau(\pi, P)$ produced by the execution of an unknown plan π that reaches the goals $G \in G[\cdot]$ starting from the initial state I in P.

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 Estimating the $P(\mathcal{O}|G)$ likelyhood with unknown domain models

Now we are ready to build an estimate of the $P(\mathcal{O}|G)$ likelyhood. Our mechanism matches the *plan recognition as planning* approach [Ramírez, 2012] except that we compute $cost(\pi^\top)$ using our compilation for *classical planning with unknown domain models*.

In more detail, we build the estimate of the $P(\mathcal{O}|G)$ likelyhood following these four steps:

1. *Build* $P^\top_G$, the classical planning problem that constrains solutions of the problem $P = \langle F, A[\cdot], s^o_0, G\rangle$ to plans $\pi^\top$ *consistent* with the input observation $\mathcal{O}(\tau)$. Note that $s^o_0 \in \mathcal{O}(\tau)$ is the initial state in the given observation.
2. *Solve* $P^\top_G$, using the proposed compilation for *classical planning with unknown domain models*. Extract from this solution (1), $cost(\pi^\top)$ (by counting the number of $\mathsf{apply}_{\xi,\omega}$ actions in the solution) but also (2), the action model A that is determined by the *insert* actions used in $\pi^\top$ to achieve the goals G.
3. *Build* $P^\perp_G$, the classical planning problem that constrains $P = \langle F, A, s^o_0, G\rangle$ to achieve $G \in G[\cdot]$ through a plan $\pi^\perp$ *inconsistent* with $\mathcal{O}(\tau)$ (where A is the set of actions extracted in step 2.).
4. *Solve* $P^\perp_G$ with a classical planner and extract $cost(\pi_\perp)$ as the length of the found solution plan.
5. Compute the $cost(\pi^\top) - cost(\pi^\perp)$ difference and plug it into equation (2) to get the $P(\mathcal{O}|G)$ likelihoods.

To compute the target probability distribution $P(G|\mathcal{O})$ plug the $P(\mathcal{O}|G)$ likelihoods into the *Bayes rule* from which the goal posterior probabilities are obtained. In this case the $P(\mathcal{O})$ probabilities are obtained by normalization (goal probabilities must add up to 1 when summed over all possible goals).

4.2 Extending the observation model of *plan recognition as planning*

The work on *plan recognition as planning* usually assumes an observation model that is referred only to logs of executed actions. However, the approach applies also to more expressive observation models that consider state observations as well, like the observation model defined above, with a simple three-fold extension:

- One fluent $\{validated_j\}_{0\leq j\leq m}$ to point at every $s^o_j \in \mathcal{O}(\tau)$ state observation.
- Adding $validated_m$ to every possible goal $G \in G[\cdot]$ to constrain 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 $s^o_j \in \mathcal{O}(\tau)$ input state observation, $(1 \leq j \leq m)$.

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

5 Evaluation

6 Related Work

The problem of *classical planning with unknown domain models* has been previously addressed [Stern and Juba, 2017]. In this work we evidence the relevance of this task for addressing *goal recognition* when the action model of the observed agent is not available (which it is typically a too strong assumption at many real-world applications).

The paper also showed that *goal recognition*, when the domain model is unknown, is closely related to the learning of planning action models. With this regard, the classical planning compilation for learning STRIPS action models [Aineto *et al.*, 2018] is very appealing because it allows to produce a STRIPS action model from minimal input knowledge (a single initial state and goals pair), and to refine this model if more input knowledge is available (e.g. observation constraints). Most of the existing approaches for learning action models aim maximizing an statistical consistency of the learned model with respect to the input observations so require large amounts of input knowledge and do not produce action models that are guaranteed to be *logically consistent* with the given input knowledge.

Our approach for *planning with an unknown domain model* is related to *goal recognition design* [Keren *et al.*, 2014].

The reason is that we are encoding the space of propositional schemes as state variables of the planning problem (the initial state encodes the *empty* action model with no preconditions and no effects) and provide actions to modify the value of this state variables as in *goal recognition design*. The aims of *goal recognition design* are however different. *Goal recognition design* applied to *goal recognition with unknown domain models* would compute the action model, in the space of possible models, that allows to reveal any of the possible goals as early as possible.

7 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.

[Slaney and Thiébaux, 2001] John Slaney and Sylvie Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.

[Stern and Juba, 2017] Roni Stern and Brendan Juba. Efficient, safe, and probably approximately complete learning of action models. In *International Joint Conference on Artificial Intelligence, (IJCAI-17)*, pages 4405–4411, 2017.