Model Recognition as Planning

Diego Aineto and **Sergio Jiménez** and **Eva Onaindia**
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València.
Camino de Vera s/n. 46022 Valencia, Spain
{dieaigar,serjice,onaindia}@dsic.upv.es

Miquel Ramírez
School of Computing and Information Systems
The University of Melbourne
Melbourne, Victoria. Australia
miquel.ramirez@nimelb.edu.au

Abstract

Given a partially observed plan execution, and a set of possible planning models (models that share the same state variables but different action models), *model recognition* is the task of identifying the model that explains the observation. The paper formalizes the *model recognition* task and introduces a novel method to estimate the probability of a STRIPS model to produce an observation of a plan execution. This method builds on top of off-the-shelf classical planning algorithms and it is robust to missing actions and intermediate states in the observation. The effectiveness of the method is shown in a set of STRIPS models encoding different kinds of *automata*. The presented approach succeeds to identify the executed automata despite the actual applied transitions, plus some state variables e.g. the internal automata state, are unobserved.

Introduction

Plan recognition is the task of predicting the future actions of an agent provided observations of its current behavior (Carberry 2001). *Goal recognition* is a closely related task that aims identifying the goals of the observed agent. Goal recognition is considered *automated planning* in reverse; while automated planning compute sequences of actions that accounts for a given goals, goal recognition compute goals that account for an observed sequence of actions (Geffner and Bonet 2013).

Diverse approaches has been proposed for plan/goal recognition such as *rule-based systems*, *parsing*, *graph-covering*, *Bayesian nets*, etc (Sukthankar et al. 2014). *Plan recognition as planning* is the model-based approach for plan/goal recognition that leverages the action model of the observed agent to compute its most likely goal (Ramírez 2012; Ramírez and Geffner 2009).

This paper formalizes the *model recognition* task where the object to recognize is not a goal but the *planning model* that shapes the behavior of the observed agent. Given a partially observed plan execution, and a set of possible planning models (models that share the same state variables but that update these variables with different action models); *model recognition* is the task of identifying the model in the set with the highest probability of producing/explaining the given observation.

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

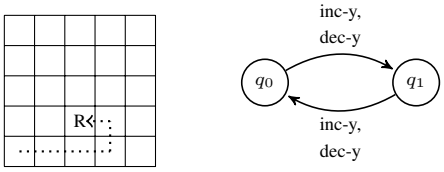

Figure 1: *(Left)* Robot navigating a 5×5 grid. *(Right)* Automata to control that the robot only increments its x-coordinate when q_0 holds (actions `inc-y` and `dec-y` update the robot y-coordinate and switch the automata state).

To better illustrate *model recognition*, imagine a robot in a $N \times N$ grid whose navigation is determined by the STRIPS model of Figure 2. According to this model, the robot can increment its *x-coordinate* when $q0$ holds (i.e. at *even rows* if $q0$ holds initially) and decrement it when $q1$ holds (at *odd rows* if $q0$ holds initially). Apart from the model of Figure 2, different models can be defined within the same state variables (e.g. altering the way $q0$ and $q1$ are required and updated) and these models can shape different kinds of robot navigation. Given a partially observed plan execution (like the one illustrated at Figure 1 where state variables $q0$ and $q1$ are unobserved) *model recognition* aims to identify the action model that produced/explains that observation.

Model recognition is of interest because once the planning model is recognized, then the model-based machinery for automated planning becomes applicable (Ghallab, Nau, and Traverso 2004). In addition, it enables identifying different kinds of automata by observing their execution. It is well-known that diverse automata representations, like *finite state controllers*, *Büchi automata*, *push-down automata*, GOLOG *programs* or *reactive policies*, can be encoded as classical planning models (Baier, Fritz, and McIlraith 2007; Bonet, Palacios, and Geffner 2010; Patrizi et al. 2011; Ivankovic and Haslum 2015; Segovia-Aguas, Jiménez, and Jonsson 2017).

The paper also introduces *model recognition as planning*; a novel method to estimate the probability of a given STRIPS model to produce an observed plan execution. Our method is built on top of off-the-shelf classical planning algorithms

```
(:action inc-x
  :parameters (?v1 ?v2)
  :precondition (and (xcoord ?v1) (next ?v1 ?v2) (q0))
  :effect (and (not (xcoord ?v1)) (xcoord ?v2)))

(:action dec-x
  :parameters (?v1 ?v2)
  :precondition (and (xcoord ?v1) (next ?v2 ?v1) (q1))
  :effect (and (not (xcoord ?v1)) (xcoord ?v2)))

(:action inc-y-even
  :parameters (?v1 ?v2)
  :precondition (and (ycoord ?v1) (next ?v1 ?v2) (q0))
  :effect (and (not (ycoord ?v1)) (ycoord ?v2)
               (not (q0)) (q1)))

(:action inc-y-odd
  :parameters (?v1 ?v2)
  :precondition (and (ycoord ?v1) (next ?v1 ?v2) (q1))
  :effect (and (not (ycoord ?v1)) (ycoord ?v2)
               (not (q1)) (q0)))

(:action dec-y-even
  :parameters (?v1 ?v2)
  :precondition (and (ycoord ?v1) (next ?v2 ?v1) (q0))
  :effect (and (not (ycoord ?v1)) (ycoord ?v2)
               (not (q0)) (q1)))

(:action dec-y-odd
  :parameters (?v1 ?v2)
  :precondition (and (ycoord ?v1) (next ?v2 ?v1) (q1))
  :effect (and (not (ycoord ?v1)) (ycoord ?v2)
               (not (q1)) (q0)))
```

Figure 2: Example of a STRIPS action model (given in PDDL) for robot navigation in a $N \times N$ grid.

and is robust to missing intermediate states and actions in the observed plan execution. We evaluate the effectiveness of *model recognition as planning* with sets of STRIPS models that represent different *automata*. All the *automata* in a set are defined within the same state variables but with different *transition functions*. We show that *model recognition as planning* identifies the executed *automata* despite the actual applied transitions, plus some state variables (e.g. the internal automata state), are unobserved.

Background

This section formalizes the models for *classical planning* and for the *observation* of the execution of a classical plan.

Classical planning with conditional effects

F is the set of *fluents* or *state variables* (propositional variables). A *literal* l is a valuation of a fluent $f \in F$, i.e. either $l = f$ or $l = \neg f$. L is a set of literals that represents a partial assignment of values to fluents, and $\mathcal{L}(F)$ is the set of all literals sets on F, i.e. all partial assignments of values to fluents. A *state* s is a full assignment of values to fluents. We explicitly include negative literals $\neg f$ in states and so $|s| = |F|$ and the size of the state space is $2^{|F|}$.

A *planning frame* is a tuple $\Phi = \langle F, A \rangle$, where F is a set of fluents and A is a set of *actions*. An action $a \in A$ is defined with *preconditions*, $\mathsf{pre}(a) \in \mathcal{L}(F)$, *positive effects*, $\mathsf{eff}^+(a) \in \mathcal{L}(F)$, and *negative effects* $\mathsf{eff}^-(a) \in \mathcal{L}(F)$. 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 $\mathsf{pre}(a) \subseteq s$. And the result of applying a in s is $\theta(s, a) = \{s \setminus \mathsf{eff}^-(a)) \cup \mathsf{eff}^+(a)\}$.

A *planning problem* is defined as a tuple $P = \langle F, A, I, G \rangle$, where I is the initial state in which all the fluents of F are assigned a value true/false and G is the goal set. A *plan* π for P is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$, and $|\pi| = n$ denotes its *plan length*. The execution of π in the initial state I 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 trajectory $\tau(\pi, P)$ that solves P is one in which $G \subseteq s_n$.

An action $a_c \in A$ with conditional effects 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 *triggered effects* resulting from the action application are the effects whose conditions hold in s:

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

The result of applying action a_c in state s is $\theta(s, a_c) = \{s \setminus \mathsf{eff}_c^-(s, a)) \cup \mathsf{eff}_c^+(s, a)\}$, where $\mathsf{eff}_c^-(s, a) \subseteq triggered(s, a)$ and $\mathsf{eff}_c^+(s, a) \subseteq triggered(s, a)$ are, respectively, the triggered *negative* and *positive* effects.

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** $\langle s_0^0, s_1^o, \ldots, s_m^o \rangle$ 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 $\forall i, s_i^0 \subseteq s_i$. In practive, 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 $[1, |F|]$. Exceptionally, s_m^o cannot be fully unobservable for the purpose of our task (we will elaborate on this issue later on).

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.

In our model recognition task, we will always assume an *empty sequence of observed actions* and so we will only work with the observed states. Figure 1 illustrates a *partial observation* of a five-state trajectory {`<(xcoord 2)(ycoord 1)>, <(xcoord 3)(ycoord 1)>, <(xcoord 4)(ycoord 1)>, <(xcoord 4)(ycoord 2)>, <(xcoord 3)(ycoord 2)>`}. Note that this observation only contains fluents of the type `(xcoord ?v)` and `(ycoord ?v)`, and the value of the remaining fluents, namely `(next ?v1 ?v2)`, `(q0)` and `(q1)`, is not observable in any of the five states.

Model Recognition

The *model recognition* task is a tuple $\langle P, M, \mathcal{O} \rangle$ where:

- $P = \langle F, A[\cdot], I, G \rangle$ is a planning problem where $A[\cdot]$ is a set of actions. For each $a \in A[\cdot]$, the semantics of a is unknown; i.e. the functions ρ and/or θ of a are undefined.
- $M = \{\mathcal{M}_1, \ldots, \mathcal{M}_k\}$ is a set of *k different planning models* for the actions in $A[\cdot]$. A model $\mathcal{M} \in M$ defines the semantics of every action in $A[\cdot]$. Planning models differ in the $\langle \rho, \theta \rangle$ functions of the actions but they all use the same set of state variables F.
- $\mathcal{O}(\tau)$ is an observation of the execution of an unknown solution plan π for the planning problem P.

Model recognition can be understood as a *classification task* where each class is represented by a different planning model $\mathcal{M} \in M$, and the observed plan execution $\mathcal{O}(\tau)$ is the single example to classify. The planning model associated to each class acts as the corresponding *class prototype* and it summarizes any observation of a plan execution that could be synthesized with such model (i.e. the set of all the examples that belong to that class). We follow the *naive Bayes classifier* to assign a model $\mathcal{M} \in M$ to a given observation $\mathcal{O}(\tau)$. The *solution* to the model recognition task is then the subset of models in M that maximizes this expression.

$$argmax_{\mathcal{M} \in M} P(\mathcal{O}|\mathcal{M}) P(\mathcal{M}). \quad (1)$$

Formulating the $P(\mathcal{O}|\mathcal{M})$ likelihood

The $P(\mathcal{M})$ probability expresses whether one model is known to be a priori more likely than the others. If this probability is not given as input it is reasonable to assume that, *a priori*, all models are equiprobable. The challenge of our formulation to the model recognition task is the definition of $P(\mathcal{O}|\mathcal{M})$, the likelihood that expresses the probability of observing $\mathcal{O}(\tau)$ when $\mathcal{M}$ is the planning model.

Our approach to formulate the $P(\mathcal{O}|\mathcal{M})$ likelihood is to assess the cost of transforming $\mathcal{M}$ into a model $\mathcal{M}'$ that produces a trajectory $\tau(\pi, P)$ such that: (1) $\tau(\pi, P)$ reaches the goals in P and (2) $\tau(\pi, P)$ is consistent with observation $\mathcal{O}(\tau)$. Figure 3 shows the *Bayesian network* that embodies this procedure. Regarding this network we have the following formulation of the $P(\mathcal{O}|\mathcal{M})$ *likelihood*:

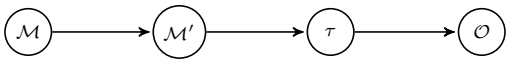

Figure 3: *Bayesian network* representing that model $\mathcal{M}$ is *transformable* into a model $\mathcal{M}'$ that produces a trajectory $\tau(\pi, P)$ that (1) reaches the goals in P and (2) it is consistent with $\mathcal{O}(\tau)$.

$$P(\mathcal{O}|\mathcal{M}) = \sum_{\mathcal{M}'} \sum_{\tau} P(\mathcal{M}'|\mathcal{M}) P(\tau|\mathcal{M}') P(\mathcal{O}|\tau), \quad (2)$$

where τ ranges over all the trajectories consistent with $\mathcal{O}(\tau)$ that can be synthesized with a model $\mathcal{M}'$ and $\mathcal{M}'$ ranges over all the models that can be generated *transforming* $\mathcal{M}$.

The exact computation of $P(\mathcal{O}|\mathcal{M})$ with equation (2) is intractable. For most planning problems the set of trajectories consistent with an arbitrary observation can easily be huge, infinite in the case of planning problems without deadends (Lesh and Etzioni 1995). Even worse, the number of models $\mathcal{M}'$ that can be generated *transforming* a given classical planning model $\mathcal{M}$ explodes combinatorially with the number of state variables. Instead, our approach is to estimate of $P(\mathcal{O}|\mathcal{M})$ using an *edit distance* referred to STRIPS planning models. *Edit distances* are similarity metrics, traditionally computed over *strings* or *graphs*, which have been proved successful for *pattern recognition* (Masek and Paterson 1980; Bunke 1997). In this work, we assess the cost of *transforming* $\mathcal{M}$ into $\mathcal{M}'$ computing *edit distances* between STRIPS planning models.

Recognition of STRIPS planning models

Our formalization of a model recognition task is valid for any planning model but our *edit distance* measure is exclusively for STRIPS models and so we restrict our attention to STRIPS–compilable planning models[1]. Thus, we focus on the recognition of $\mathcal{M} \in M$, where M is a set of STRIPS planning models and $\mathcal{M}$ defines the semantics of the actions through a set of STRIPS action schemata.

Well-defined STRIPS action schemata

STRIPS action schemata provide a compact representation for specifying classical planning models. Figure 2 shows six STRIPS action schemata that shape a particular kind of robot navigation in $N \times N$ grids (no matter the grid size).

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.

Definition 1 (Comparable STRIPS action schemata)**.** *Two* STRIPS *schemata* ξ *and* ξ' *are* **comparable** *iff* $pars(\xi) =$

[1] An edit distance for other planning models is also definable

$pars(\xi')$*, i.e, both share the same list of parameters.*[2]

For instance, we claim that the six action schemata of Figure 2 are *comparable* while, for example, the `stack(?v1,?v2)` and `pickup(?v1)` schemata from a four operator *blocksworld* (Slaney and Thiébaux 2001) are not. Last but not least, we say that two STRIPS models $\mathcal{M}$ and $\mathcal{M}'$ are *comparable* iff there exists a bijective function $\mathcal{M} \mapsto \mathcal{M}^*$ that maps every action schema $\xi \in \mathcal{M}$ to a comparable schemata $\xi' \in \mathcal{M}'$ and vice versa.

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 $pars(\xi)$. We denote this set of FOL interpretations as $\mathcal{I}_{\Psi,\xi}$. For any of the six action schemata of Figure 2, the $\mathcal{I}_{\Psi,\xi}$ set contains the same ten elements, $\mathcal{I}_{\Psi,\xi}$ ={`xcoord(`v_1`)`, `xcoord(`v_2`)`, `ycoord(`v_1`)`, `ycoord(`v_2`)`, `q0()`, `q1()`, `next(`v_1`,`v_1`)`, `next(`v_1`,`v_2`)`, `next(`v_2`,`v_1`)`, `next(`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 constrained by a set $\mathcal{C}$ that includes:

- *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 $2^{2\times|\mathcal{I}_{\Psi,\xi}|}$. For every action schema in the navigation model of Figure 2 then $2^{2\times 10} = 1,048,576$.
- *Domain-specific constraints*. One can introduce domain-specific knowledge to constrain further the space of possible schemata. For instance, in a *robot navigation* model like the one in Figure 2, `q0()` and `q1()` are exclusive so they cannot hold at the same time in a $pre(\xi)$/$del(\xi)$/$add(\xi)$ list. Further, `next(`v_1`,`v_1`)` and `next(`v_2`,`v_2`)` will not appear in any of these lists because the `next` predicate codes the *successor* function for *natural numbers*. These domain-specific constraints reduce further the size of the space of possible action schemata to $2^{2\times 7} = 16,384$ (for every schema in the navigation model of Figure 2).

Definition 2 (Well-defined STRIPS action schemata)**.** *Given a set of* predicates Ψ*, a list of action* parameters $pars(\xi)$*, and set of FOL constraints $\mathcal{C}$, ξ is a* **well-defined STRIPS action schema** *iff its three lists $pre(\xi) \subseteq \mathcal{I}_{\Psi,\xi}$, $del(\xi) \subseteq \mathcal{I}_{\Psi,\xi}$ and $add(\xi) \subseteq \mathcal{I}_{\Psi,\xi}$ only contain elements in $\mathcal{I}_{\Psi,\xi}$ and they satisfy all the constraints in $\mathcal{C}$.*

We say a planning model $\mathcal{M}$ is *well-defined* if all its STRIPS action schemata are *well-defined*.

[2]In STRIPS models, $pars(\xi) = pars(\xi')$ implies the number of parameters must be the same. For other planning models that allow object typing, the equality implies that parameters share the same type

Edit distances for STRIPS planning models

First, we define the two edit *operations* on a schema ξ that belongs to a STRIPS model $\mathcal{M} \in M$:

- *Deletion*. Given $\xi \in \mathcal{M}$, an element from any of the lists $pre(\xi)$/$del(\xi)$/$add(\xi)$ is removed such that the result is a *well-defined* STRIPS action schema.
- *Insertion*. Given $\xi \in \mathcal{M}$, an element in $\mathcal{I}_{\Psi,\xi}$ is added to any of the lists $pre(\xi)$/$del(\xi)$/$add(\xi)$ such that the result is a *well-defined* action schema.

We can now formalize an *edit distance* that quantifies how similar two given STRIPS models are. The distance is symmetric and meets the *metric axioms* provided that the two edit operations, *deletion* and *insertion*, have the same positive cost.

Definition 3 (Edit distance)**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two* comparable *and* well-defined STRIPS *planning models within the same set of predicates Ψ. The* **edit distance** *$\delta(\mathcal{M}, \mathcal{M}')$ is the minimum number of* edit operations *that is required to transform $\mathcal{M}$ into $\mathcal{M}'$.*

Since $\mathcal{I}_{\Psi,\xi}$ is a bounded set, the maximum number of edits that can be introduced to an action model is bounded as well. The **maximum edit distance** of a STRIPS model $\mathcal{M}$ built with predicates Ψ is $\delta(\mathcal{M}, *) = \sum_{\xi\in\mathcal{M}} 3 \times |\mathcal{I}_{\Psi,\xi}|$ (note that if we consider the set of syntactic constraints then $\delta(\mathcal{M}, *) = \sum_{\xi\in\mathcal{M}} 2 \times |\mathcal{I}_{\Psi,\xi}|$).

An observation of the execution of a plan generated with an action model $\mathcal{M}$ further constraints the space of possible action schemata of $\mathcal{M}$. The *semantic knowledge* included in the observations introduce a third type of constraints, that we will call *observation constraints*, and that can be added to the set $\mathcal{C}$. In addition, *observation constraints* allow us to define an edit distance to elicit the value of $P(\mathcal{O}|\mathcal{M})$. It can be argued that the shorter this distance the better the given model explains the given observation.

Definition 4 (Observation edit distance)**.** *Given a planning problem P, an observation $\mathcal{O}(\tau)$ of the execution of a plan that solves P and a* STRIPS *planning model $\mathcal{M}$ (all defined within the same set of predicates Ψ). The* **observation edit distance***, $\delta^o(\mathcal{M}, \mathcal{O})$, is the minimal edit distance from $\mathcal{M}$ to any* comparable *and well-defined model $\mathcal{M}'$ s.t. $\mathcal{M}'$ produces a trajectory $\tau(\pi, P)$ that reaches the goals in P and is* consistent *with $\mathcal{O}(\tau)$;*

$$\delta^o(\mathcal{M}, \mathcal{O}) = \min_{\forall \mathcal{M}' \to \mathcal{O}} \delta(\mathcal{M}, \mathcal{M}')$$

$\delta^o(\mathcal{M}, \mathcal{O})$ can also be defined through the edition that the observation $\mathcal{O}(\tau)$ requires to fit $\mathcal{M}$. This implies defining *edit operations* that modify the observation $\mathcal{O}(\tau)$ instead of the model $\mathcal{M}$ (Yang, Wu, and Jiang 2007; Sohrabi, Riabov, and Udrea 2016). Our definition of *observation edit distance* is more practical since the size of $\mathcal{I}_{\Psi,\xi}$ is usually much smaller than F (the number of variables in the action schemata should normally be lower than the number of objects in a planning problem).

Definition 5 (*Closest consistent models*)**.** *Given a model $\mathcal{M}$, the set M^* of the* **closest consistent models** *is the set*

of models $\mathcal{M}'$ that: (1) produce a trajectory $\tau(\pi, P)$ that reaches the goals in P and is consistent *with $\mathcal{O}(\tau)$ and (2) their* edit distance *to $\mathcal{M}$ is minimal;*

$$\underset{\forall \mathcal{M}' \rightarrow \mathcal{O}}{\arg\min}\, \delta(\mathcal{M}, \mathcal{M}')$$

Approximating the $P(\mathcal{O}|\mathcal{M})$ *likelihood*

Now we we are ready to formulate an informative estimate of the $P(\mathcal{O}|\mathcal{M})$ likelihood for the particular case where models $\mathcal{M}$ are specified with STRIPS action schemata.

Full observability of the executed plan. The *full observability of the executed plan* is a too strong assumption for *model recognition* but it allows us to understand how to build a reasonable estimate of $P(\mathcal{O}|\mathcal{M})$ for the general case.

Under the assumption of *full observability*, there is only a single possible trajectory $\tau^*(\pi, P)$ *consistent* with the input observation so $P(\mathcal{O}|\tau^*) = 1$. Further, there is also a single model that can exactly produce that trajectory (otherwise models are identical, at least, in the actions relevant to the observed trajectory so they can be considered the same model).

Provided that there is a single possible trajectory and a single possible model consistent with the input observation, i.e. $M^* = \{\mathcal{M}^*\}$ then, the probabilities of expression (2) are not added up and expression (1) simplifies to:

$$argmax_{\mathcal{M} \in M} P(\mathcal{M}^*|\mathcal{M}) P(\mathcal{M}). \tag{3}$$

Note that the term $P(\tau|\mathcal{M}^*)$ is taken out of the maximization because is independent of the input model $\mathcal{M} \in M$.

Partial observability of the executed plan. In a similar way, an approximation to $P(\mathcal{O}|\mathcal{M})$ can be built for the general case, where the executed plan is partially observed. We add the following two assumptions to deal with this general case:

1. The *principle of rationality*: The expectation that intentional agents will tend to choose actions that achieve their goals most efficiently, given their knowledge of the actual world state (Dennett 1983).
2. *The best plans for P are unique or have different cost.*

Similar assumptions were taken in previous work for *goal recognition* (Ramírez 2012). They allow us to assume that the sum in equation (2) is dominated by its largest term, so other terms in the sum are not added up. The largest term corresponds here to the shortest trajectory τ^* consistent with the observation (which now is assumed to be unique). Note that the more complete is the observation of the plan execution the more accurate is our estimate because it becomes more likely to assume that there is only one trajectory consistent with the input observation.

Again, we have that there is a single model $\mathcal{M}^*$ that can produce a given trajectory τ^*. With this said, both $P(O|\tau^*)$ and $P(\tau^*|\mathcal{M}^*)$ are independent of $\mathcal{M} \in M$ so equation (1) simplifies once again to equation (3) under the previous two assumptions.

The $P(\mathcal{M}^*|\mathcal{M})$ probability distribution. $P(\mathcal{M}'|\mathcal{M})$ indicates the probability of *transforming* a classical planning model $\mathcal{M}$ into a model $\mathcal{M}'$ by exclusively using the two *edit operations* previously defined, *deletion* and *insertion*.

We know from *pattern recognition* (Devroye, Györfi, and Lugosi 2013) that, if string symbols are uniformly random and independent, the distance of a given string to a fixed string follows a *Binomial distribution*. Moreover, the probability that a particular string will be within a distance D is the Cumulative Distribution Function (CDF) for the *Binomial distribution* (Wilcox 1981):

$$CDF(D) = \sum_{d=0}^{D} \binom{N}{d} p^d (1-p)^{N-d} \tag{4}$$

where N is the length of the string, and $p < 0.5$ since we consider that the cost of applying an *edit operation* is higher than do not applying it.

With this regard, and considering that STRIPS models $\mathcal{M} \in M$ can be encoded with a propositional representation of fixed length N, we formulate the $P(\mathcal{M}'|\mathcal{M})$ probability distribution mapping the distance $\delta(\mathcal{M}, \mathcal{M}')$ according to equation:

$$P(\mathcal{M}'|\mathcal{M}) = p^d (1-p)^{N-d} \tag{5}$$

where $d = \delta(\mathcal{M}, \mathcal{M}')$. Likewise we can formulate $P(\mathcal{M}^*|\mathcal{M})$ with this same equation (5) but instead, the parameter d is given by the *observation distance*, $d = \delta^o(\mathcal{M}, \mathcal{O})$.

In other words, we are modeling the edition of a STRIPS planning model as a *Bernoulli process* in which there is a sequence of N independent events representing N binary decisions (the N possible applications of the edition operations) such that for every of these events $P(X = \top) = p$ and $P(X = \bot) = 1 - p$.

Model Recognition as planning

This section shows that $\delta^o(\mathcal{M}, \mathcal{O})$, and hence an approximation to $P(\mathcal{O}|\mathcal{M}))$, can be computed with a compilation-to-planning approach as the one proposed in (Aineto, Jiménez, and Onaindia 2018) (AJO approach hereafter) for learning STRIPS models. The AJO approach receives as input an empty model $\mathcal{M}$, which only contains the headers of the action schemata formed of $\xi = \langle name(\xi), pars(\xi) \rangle$, and an observation of a plan execution $\mathcal{O}(\tau)$ (extensible to a set of observations), and it returns a model $\mathcal{M}'$ with specification of preconditions and effects of each action schema included in $\mathcal{M}$ such that the validation of $\mathcal{O}(\tau) = \langle s_0^o, a_1^o, s_1^o \ldots, a_l^o, s_m^o \rangle$ following $\mathcal{M}'$ is successful; i.e., it holds $\rho(s_{i-1}, a_i)$ for every observed action of $\mathcal{O}(\tau)$ and $s_i = \theta(s_{i-1}, a_i)$ for every observed state of $\mathcal{O}(\tau)$.

In essence, the task $\langle \mathcal{M}, \mathcal{O} \rangle$ of the AJO approach – learning $\mathcal{M}$ that satisfies $\mathcal{O}$– can be interpreted as editing the empty action schemata of $\mathcal{M}$ introducing preconditions and effects until $\mathcal{O}$ is validated with the resulting model. We leverage the same idea to compute $\delta^o(\mathcal{M}, \mathcal{O})$ with the exception that now our $\mathcal{M}$ is not empty but $\mathcal{M} = \{\xi_1, \ldots, \xi_m\}$, where $\xi_i =$

$\langle name(\xi_i), pars(\xi_i), pre(\xi_i), add(\xi_i), del(\xi_i)\rangle$, $1 \leq i \leq m$.

Editing/learning the action schemata of $\mathcal{M}$ in the AJO approach is addressed converting the task into a classical planning problem, which is later solved with a planner. The intuition behind this compilation is that a solution plan to the problem is a sequence of: (a) *edit actions* on the schemata of $\mathcal{M}$ to build $\mathcal{M}'$ and (b) *validate actions* that apply $\mathcal{M}'$ in $\mathcal{O}(\tau)$. The adaptation of this compilation scheme for solving $\delta^o(\mathcal{M}, \mathcal{O})$ results in a planning problem $P' = \langle F', A', I', G'\rangle$ whose objective is to determine the preconditions and effects that need to be added or deleted to the action schemata of $\mathcal{M}$ so as to satisfy $\mathcal{O}$. The accomplishment of this task requires therefore a propositional encoding of the components of the action schemata. Specifically:

- F' contains the necessary fluents to represent the pre, add and del of the action schemata. It is a set of *editable fluents* of the type $\{pre\_e\_\xi, del\_e\_\xi, add\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ 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 $pars(\xi)$. Additionally, F also contains fluents to encode the *observation constraints*; that is, fluents to iterate through the l observed actions and m observed states of $\mathcal{O}(\tau)$.
- A' comprises two types of actions with conditional effects:
  - actions for editing $\xi \in \mathcal{M}$ that follow the *syntactic constraints* for well-defined STRIPS action schemata. Hence, A' contains actions for inserting a new precondition pre_e_ξ or a new positive effect add_e_ξ in ξ when neither pre_e_ξ, del_e_ξ or add_e_ξ exist in ξ, and actions for inserting a new negative effect when neither del_e_ξ or add_e_ξ appear in ξ but pre_e_ξ does.
  - actions for applying the new action schemata of the edited model $\mathcal{M}'$ and validating the observed states of $\mathcal{O}(\tau)$. Particularly, the *apply* actions check the preconditions and produce the effects defined by the *editable fluents*.
- I' encodes the editable fluents pre_e_ξ, del_e_ξ and add_e_ξ that hold in the action schemata of $\mathcal{M}$.
- G' contains the necessary fluents to check that all the observed actions and states of $\mathcal{O}(\tau)$ are generated correctly following the edited model $\mathcal{M}'$.

We now show an example of a solution plan to a planning problem P' that results from compiling a specific task $\langle \mathcal{M}, \mathcal{O}\rangle$. Consider that $\mathcal{M}$ is the model of Figure 2 except that the schema `inc-x` is defined without preconditions and its positive/negative effects are swapped wrt Figure 2. And consider an observation that only contains the initial and final state of Figure 1; i.e., $\mathcal{O}(\tau) = \langle s_0^o, s_m^o\rangle = \langle (1,1), (3,2)\rangle$. The plan found by a planner to P' is shown in figure 4. The first seven steps are the edit actions to *fix* the schema `inc-x`; step 07 is a validate action that sets the robot in the initial state $s_0^o = (1,1)$; step 08 applies the action `(inc-x 1 2)` and moves the robot one cell to the right to $(2,1)$; steps 09-11 also move the robot one cell to the right;

```
00 : (insert_pre_xcoord_v1_inc-x)    07 : (validate_0)
01 : (insert_pre_next_v1_v2_inc-x)   08 : (apply_inc-x 1 2)
02 : (insert_pre_q0_inc-x)           09 : (apply_inc-x 2 3)
03 : (delete_del_xcoord_v2_inc-x)    10 : (apply_inc-x 3 4)
04 : (delete_add_xcoord_v1_inc-x)    11 : (apply_inc-y-even 1 2)
05 : (insert_del_xcoord_v1_inc-x)    12 : (apply_dec-x 4 3)
06 : (insert_add_xcoord_v2_inc-x)    13 : (validate_1)
```

Figure 4: A plan for $\langle \mathcal{M}, \mathcal{O}\rangle$ where $\mathcal{M}$ is a modification of the model of Figure 2 and $\mathcal{O}$ a partial observation from Figure 1.

step 11 applies an action that increases coordinate y to an even number, thus moving robot to row 2; step 12 moves the robot one cell to the left and, finally, action `(validate_1)` checks the robot position is consistent with the final state $s_m^o = (3,2)$.

The value of the *observation distance* $\delta^o(\mathcal{M}, \mathcal{O})$ is given by the number of *edit operations* (insertions and deletions) required by $\mathcal{M}$ to be validated in the input observation. In the case of the above example, the distance equals 7.

Evaluation

This section evaluates the empirical performance of *model recognition as planning*. To do so, we define a set M of different planning models, that share the same state variables but that update these variables according to different action schemata. Then for each model $\mathcal{M} \in M$, we generate *partial observations* $\mathcal{O}(\tau)$ of *plan trajectories* that are computed with that model. Finally, we apply our *model recognition as planning* method to identify which model, among on the models in M, was actually used for generating a given $\mathcal{O}(\tau)$ observation. For all the experiments, we assume equal prior probabilities, meaning $P(\mathcal{M}) = \frac{1}{|M|}$ for every $\mathcal{M} \in M$.

For the purpose of the experiments below, we introduce a particular class of $\mathcal{O}(\tau)$ observations. This new class allows us to distinguish between observable data, which may come from sensors, and hidden or latent factors that cannot be observed.

Definition 6 (Φ-observation)**.** *Given a subset of fluents $\Phi \subseteq F$ we say that $\mathcal{O}(\tau)$ is a Φ-observation of the execution of π on P iff, for every $1 \leq i \leq m$, each observed state s_i^o only contains fluents in Φ.*

Reproducibility. All experiments were run in an Intel Core i5 3.10GHz x 4 16GB of RAM and the classical planner we used to solve the instances that result from the compilation was MADAGASCAR (Rintanen 2014) because of its ability to deal with classical planning problems with dead-ends (López, Celorrio, and Olaya 2015).

The compilation source code, evaluation scripts and benchmarks (including the used training and test sets) are fully available at this anonymous repository so any experimental data reported in the paper can be reproduced.

Recognition of *Regular Automata*.

The first experiment, which doubles as a proof of concept, leverages *model recognition as planning* to perform a typical *string classification* problem. In this experiment, the system receives (1) the string to classify (the $\mathcal{O}(\tau)$ observation) and

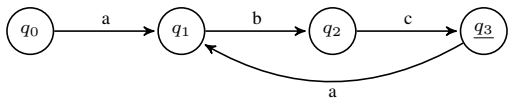

Figure 5: Three-symbol four-state *regular automata* for recognizing the $(abc)^+$ language ($\underline{q_3}$ is the acceptor state).

(2), a set M of different planning models, where each $\mathcal{M} \in M$ codes a different class (i.e. a different regular automata that accepts any string that belong to the class).

Figure 5 illustrates a three-symbol four-state *regular automata* for recognizing the $(abc)^+$ language. The *input alphabet* is $\Sigma = \{a, b, c\}$, and the machine states are $Q = \{q_0, q_1, q_2, \underline{q_3}\}$ (where $\underline{q_3}$ is the only acceptor state). For instance, the execution of the planning model coding the *regular automata* defined in Figure 5, and with the input string $abcabc$, produces the following six-action plan $(a, q_0 \rightarrow q_1)$, $(b, q_1 \rightarrow q_2)$, $(c, q_2 \rightarrow q_3)$, $(a, q_3 \rightarrow q_1)$, $(b, q_1 \rightarrow q_2)$, $(c, q_2 \rightarrow q_3)$.

Note that the the string to classify is a Φ-*observation* since it does not contain any information about the structure of the possible regular automata. This means that $\mathcal{O}(\tau)$ is a Φ-observation where fluents representing the internal machine state are unknown and the applied transitions (actions) are unobserved.

In this experiment, M comprises 5 planning models representing different regular automata defined with 5 states and a 4-symbol alphabet. The regular languages defined by these five automata are the following:

- $\mathcal{L}1$: $a^+(b|c)d(dd)^*a$
- $\mathcal{L}2$: $bd(abd)^*cd^*c^+$
- $\mathcal{L}3$: $d^*c(ac)^*db^*ac^*b^+$
- $\mathcal{L}4$: $(cc)^+bd(abd)^*$
- $\mathcal{L}5$: $(d|a)a(ba)^*c^+d(dc^*d)^*$

For each regular language, we generated 20 random strings (observations) of length 20 to 30 for a total of 100 strings (observations). Table 1 shows the confusion matrix resulting from classifying these 100 strings with *model recognition as planning*. In this matrix, rows represent the actual class and columns the class predicted by our *model recognition as planning* approach. Even though we are using a suboptimal classical planner, we can see that the class for all the 100 input strings was correctly recognized. These results proof that *model recognition as planning* is suitable for classification tasks where classes are given as generative planning models.

	$\mathcal{L}1$	$\mathcal{L}2$	$\mathcal{L}3$	$\mathcal{L}4$	$\mathcal{L}5$
$\mathcal{L}1$	20	0	0	0	0
$\mathcal{L}2$	0	20	0	0	0
$\mathcal{L}3$	0	0	20	0	0
$\mathcal{L}4$	0	0	0	20	0
$\mathcal{L}5$	0	0	0	0	20

Table 1: Confusion matrix for *regular automata recognition*.

Recognition of failures in *blocksworld* domain

For our second experiments we have generated a set of observations for the *blocksworld* domain where some of the actions may fail. In more detail, the failures we consider are:

- Stack fails and does nothing.
- Unstack fails and does nothing.
- Unstack fails and drops the block on the table.

We limited the errors to one type of error per observation, but it would be possible to recognize combinations of errors. The input models extend a 4-schemata *blocksworld* domain with an additional action schema that encodes one of the above failures. This means that the failure can be identified by finding the model that better explains the observation.

Our aim with this experiment is to validate the assumptions underlying our approximation of the $P(\mathcal{O}|\mathcal{M})$ likelihood.

Recognition of navigation models.

In this experiment the planning models given in M represent different navigation models for $N \times N$ grids. In particular, we took a *classical navigation model* with actions `up, down, right, left` (to move one cell in each of these four directions) and use it to define 8 different navigation policies with respect to these additional state variables {`(q0), (q1), (min ?v), (max ?v)`}. An example is the navigation policy, shown in Figure 2. This policy allows to move right when q_0 holds while it allows to move left when q_1 holds, producing a zig-zag pattern when navigating for visiting all the cells of $N \times N$ grids.

On the other hand we generated observations from 8 different trajectories, one for each of the 8 previous planning models. The trajectories depict the path followed by each navigation model to solve the planning problem of *visiting all* cells in a 5×5 grid. As in the previous experiments, observations contain no actions and here, for each observed state, they only contain the value of the fluents encoding the *x and y coordinates* of the agent are known (i.e. $\Phi =$ {`xcoord, ycoord`}).

Again, this experiment evaluates the accuracy of *model recognition as planning* to identify the model that generated each observation under different degrees of partial observability. The novel aspect of this experiment is that now, we do not guarantee that observed states contain at least one literal, meaning that some intermediate states may be missing. Therefore, in this new scenario more than a single action may be required to reach one observed state from another.

Another interesting aspect of this experiment is that all navigation policies are at a maximum distance of just 4 editions from the base *classical navigation model* that can move in any direction at any given state. This means that, for any given model and observation, we will need to perform a maximum of 4 editions to the *classical navigation model* that can explain any observation. This aspect heavily constrains the discriminating power of the $P(\mathcal{O}|\mathcal{M})$ likelihood, and added to the low observability of the states produces a very benchmark for *model recognition as planning*.

Figure 6 shows the classification *accuracy* achieved by *model recognition as planning* with respect to a range of different degrees of observability, from 0% to 100% (full observability) with 10% increments. In this experiment we included the 0% case which corresponds to observations where only the initial and final states are observed. The figure shows that for 0% observability we were unable to unmistakably identify the generating navigation models, but starting from 10% we start to correctly classify most of the observations. Accuracy stabilizes in the range 0.875 to 1 which means that, at most, only 1 observation of the 8 was not correctly classified.

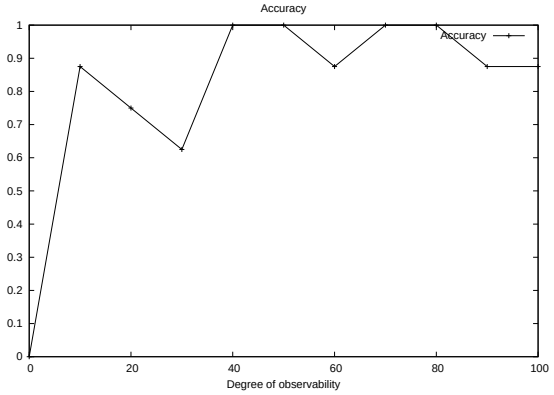

Figure 6: Classification accuracy for the recognition of navigation models.

Conclusions

This paper formalized the *model recognition* task and proposed, *model recognition as planning*, a method built on top of off-the-shelf classical planning algorithms to estimate the probability of a STRIPS model to produce a partial observation of a plan execution. The paper shows the effectiveness of *model recognition as planning* in a set of STRIPS models encoding different kinds of *automata*. *Model recognition as planning* succeeds to identify the executed automata despite the internal machine state or actual applied transitions, are unobserved.

Previous work on the learning of STRIPS action models also defined semantic error metrics to quantify the errors of a model with respect to the observation of a plan execution (Yang, Wu, and Jiang 2007). Our approach for quantifying this error is based on the definition of a edit distance for the model which allow us to not accumulate the repetition of errors coming from the the same model flaw. A related approach is recently followed for *model reconciliation* (Chakraborti et al. 2017) where model edition is used to conform the PDDL models of two different agents. Also related to this paper is the work on *Goal Recognition Design* but in that work the action model is given in advance and fixed (Keren, Gal, and Karpas 2014).

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS action models with classical planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, 399–407. AAAI Press.

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *International Conference on Automated Planning and Scheduling, (ICAPS-07(*, 26–33.

Bonet, B.; Palacios, H.; and Geffner, H. 2010. Automatic derivation of finite-state machines for behavior control. In *National Conference on Artificial Intelligence, (AAAI-10)*.

Bunke, H. 1997. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters* 18(8):689–694.

Carberry, S. 2001. Techniques for plan recognition. *User Model. User-Adapt. Interact.* 11(1-2):31–48.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *International Joint Conference on Artificial Intelligence, (IJCAI-17)*, 156–163.

Dennett, D. C. 1983. Intentional systems in ethology: The "Plangossian paradigm" defended. *Behavioral and Brain Sciences* 6(3):343–355.

Devroye, L.; Györfi, L.; and Lugosi, G. 2013. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

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

Ivankovic, F., and Haslum, P. 2015. Optimal planning with axioms. In *International Joint Conference on Artificial Intelligence, IJCAI-15*.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *International Conference on Automated Planning and Scheduling, (ICAPS-14)*.

Lesh, N., and Etzioni, O. 1995. A sound and fast goal recognizer. In *International Joint Conference on Artificial Intelligence, (IJCAI-95)*, volume 95, 1704–1710.

López, C. L.; Celorrio, S. J.; and Olaya, Á. G. 2015. The deterministic part of the seventh international planning competition. *Artificial Intelligence* 223:82–119.

Masek, W. J., and Paterson, M. 1980. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.* 20(1):18–31.

Patrizi, F.; Lipovetzky, N.; De Giacomo, G.; and Geffner, H. 2011. Computing infinite plans for ltl goals using a classical planner. In *International Joint Conference on Artificial Intelligence, (IJCAI-11)*, 2003–2008.

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

Ramírez, M. 2012. *Plan recognition as planning*. Ph.D. Dissertation, Universitat Pompeu Fabra.

Rintanen, J. 2014. Madagascar: Scalable planning with SAT. In *International Planning Competition, (IPC-2014)*.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2017. Generating Context-Free Grammars using Classical Planning. In *International Joint Conference on Artificial Intelligence, (IJCAI-17)*. AAAI Press.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan recognition as planning revisited. In *International Joint Conference on Artificial Intelligence, IJCAI-16*, 3258–3264.

Sukthankar, G.; Goldman, R. P.; Geib, C.; Pynadath, D. V.; and Bui, H. 2014. *Plan, Activity, and Intent Recognition: Theory and Practice*. Morgan Kaufmann.

Wilcox, R. R. 1981. A review of the beta-binomial model and its extensions. *Journal of Educational Statistics* 6(1):3–32.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence* 171(2-3):107–143.