Model Recognition as Planning

Diego Aineto, Sergio Jiménez, 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@unimelb.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 schemata), *model recognition* is the task of identifying the model that explains the observation. The paper formalizes this task and introduces a novel method that estimates 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 tested in three experiments, each encoding a set of different STRIPS models and all using empty-action observations: (1) a classical string classification task; (2) identification of the model that encodes a failure present in an observation; and (3) recognition of a robot navigation policy.

Introduction

Addressing the task of recognizing the mission (plans and goals) of a planning agent from a set of given observations has recently captured much attention. Rule-based systems, parsing, graph-covering or Bayesian nets have been proposed for plan/goal recognition tasks (Sukthankar et al. 2014). *Plan recognition as planning* is the model-based approach that leverages the action model of the observed agent to compute its most likely goal and predict its future actions (Ramírez and Geffner 2009; Ramírez and Geffner 2010). This paper presents the novel task of *model recognition*, where the target of the recognition is not a goal or a plan but the model that shapes the behaviour of the observed agent. Given a *partially observed plan execution*, and a *set of possible planning models*, model recognition is the task of identifying the model in the set with the highest probability of producing the given observation. Model recognition enables the identification of the agent's behaviour without need of being aware of the agent's intention.

As an example, imagine you are given a set of models, each one encoding a different navigation policy for a $N \times N$ grid. Figure 1 shows an example of one such model, according to which, the agent can only move right when it is at an even row ($q0$ holds), and left when it is at an odd row ($q1$ holds). Given the partially observed plan execution of Figure 2, where variables $q0$ and $q1$ are unobserved, model recognition aims to identify the navigation policy (model) that better explains this observation.

```
(: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 1: Example of an action schemata (given in PDDL) for robot navigation in a $N \times N$ grid.

Our proposal to tackle the task of model recognition is to compute, for each model $\mathcal{M}$ in the set, a model $\mathcal{M}'$ by editing $\mathcal{M}$ with the least needed amount of adjustments that are required for $\mathcal{M}'$ to explain the observation. We apply an *edit distance* that determines the number of insertions and deletions of preconditions and effects that are required to transform $\mathcal{M}$ into $\mathcal{M}'$. With this in mind, model recognition resembles other tasks like *model reconciliation*, which uses model editing to conform two PDDL models, where one of them can be an annotated model or an empty model, with respect to a fully observed optimal plan computed with one of the two models (Chakraborti et al. 2017; Sreedharan, Chakraborti, and Kambhampati 2018; Chakraborti, Sreedharan, and Kambhampati 2018). Also,

the *model drift* (Bryce, Benton, and Boldt 2016) approach estimates the error between a computational model, and a ground-truth model which changes as the domain evolves, as the normalized symmetric difference of the propositions that appear in the preconditions and effects of both models. In this case, plan observations are used to hypothesize variations of the model.

Unlike model reconciliation and model drift, our proposal computes the model $\mathcal{M}'$ from *partial observations* of plan executions that contain no actions and incomplete – even empty– intermediate states. Moreover, we leverage the two-model edit distance to elaborate a probabilistic distribution of a model $\mathcal{M}$, which is used to posit the model recognition task as a *classification task* with as many classes as input models. This way, we account for the possibly different prior probability of each model, a feature that cannot be considered with purely distance-based metrics.

One last consideration is related to the use of an action model learning approach with partial observability to induce the model $\mathcal{M}'$. While most approaches can learn from empty or partially specified models (Yang, Wu, and Jiang 2007; Amir and Chang 2008; Cresswell, McCluskey, and West 2013), model recognition requires the ability of editing a possibly inconsistent model $\mathcal{M}$ so that the resulting model $\mathcal{M}'$ is consistent with a partial observation in which all or many of the actions are probably missing. This poses the problem of *filling the gaps* of the incomplete observation and so it justifies using an approach to *learning action models as planning* (Aineto, Jiménez, and Onaindia 2018). Otherwise, in the case of full observability, our formalization might as well use any model editing approach such as model reconciliation or model drift.

We provide a formalization of model recognition that is applicable to any planning model but we restrict our attention to STRIPS−compilable models. It is well-known that diverse automata representations like finite state controllers, GOLOG programs or reactive policies can be encoded as classical planning models (Baier, Fritz, and McIlraith 2007; Bonet, Palacios, and Geffner 2010; Segovia-Aguas, Jiménez, and Jonsson 2018; Segovia-Aguas, Celorrio, and Jonsson 2019). As for the evaluation, the effectiveness of our approach is tested in three experiments, each encoding a set of different STRIPS models and all using empty-action observations: (1) a proof-of-concept classification task; (2) a simulation of a non-deterministic *blocksworld* under different degrees of observability; and (3) a navigation task where observations contain an unbounded number of unobservable states.

Background

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 $2^{|F|}$.

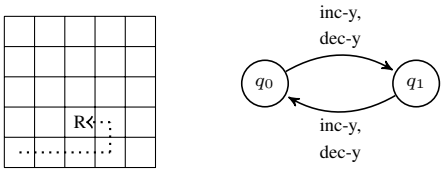

Figure 2: (Left) Robot navigating a 5×5 grid. (Right) Automata to control that the robot only increments its x-coordinate when q_0 holds (actions `inc-y` and `dec-y` update the robot y-coordinate and switch the automata state).

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*, $\mathsf{pre}(a) \in \mathcal{L}(F)$, and *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 $\theta(s, a) = \{s \setminus \neg\mathsf{eff}(a)) \cup \mathsf{eff}(a)\}$, with $\neg\mathsf{eff}(a) = \{\neg l : l \in \mathsf{eff}(a)\}$.

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 $s_i = \theta(s_{i-1}, a_i)$. The *trajectory length* of $\tau(\pi, P)$ is given by the plan length of π. 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 $a_c \in A$ is applicable in a state s if and only if $\mathsf{pre}(a_c) \subseteq s$, 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 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 $\tau(\pi, P)$. 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.

Figure 2 illustrates a partial observation of a six-state trajectory $\langle\{$`(xcoord 0)(ycoord 0)`$\},\{$`(xcoord 1)(ycoord 0)`$\},\ldots,\{$`(xcoord 2)(ycoord 1)`$\}\rangle$. This observation only contains fluents of the predicates `(xcoord ?v)` and `(ycoord ?v)`, and the value of the remaining fluents, corresponding to predicates `(next ?v1 ?v2)`, `(q0)` and `(q1)`, is unobservable in the six 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 a trajectory $\tau(\pi, P)$ produced by the execution of an unknown plan π that solves 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.

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

The $P(M)$ probability expresses whether one model is known to be a priori more likely than the others. When this probability is not given as input, we can reasonable assume that, a priori, all models are equiprobable. This is precisely what distance-based model edits systems assume, that all the input models are equiprobable, because this kind of prior knowledge is not exploitable when using only distances as classification metric. Consequently, a probability-based formulation allows for a more general definition of a Model Recognition framework.

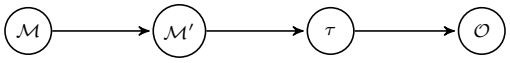

Figure 3: Bayesian network representing that model $\mathcal{M}$ is transformable into a model $\mathcal{M}'$ that produces a trajectory $\tau(\pi, P)$ that (1) reaches the goals in P and (2) it is consistent with $\mathcal{O}(\tau)$.

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

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*:

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

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 $P(\mathcal{O}|\mathcal{M})$ using an *edit distance* defined for 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

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

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 1 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 a list of *parameters* $pars(\xi)$, and three lists 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* $pars(\xi) = pars(\xi')$*, i.e, both share the same list of parameters*[2].

For instance, we claim that the six action schemata of Figure 1 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 Ψ be the set of *predicates* that shape the propositional state variables F. 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 1, 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`)`}.

Although 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 1 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 1, `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 1).

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*.

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 schema 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 $\mathcal{M}$ 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*

[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.

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 editing 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;*

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

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

Now 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 it 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. Similarly as the rationality principle (Dennett 1983) applied to agents acting rationally, we assume that agents will perform the least needed amount of adjustments to its model needed to explain the observation.
2. A model can only produce one trajectory consistent with the given observation $\mathcal{O}$ and any such trajectory is equally likely. As future work we plan to relax this assumption by weighting the edit distance against the trajectory length.

These assumptions mean 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 closest consistent model $\mathcal{M}^*$. Note that the more complete the observation of the plan execution is the more accurate our estimate becomes, because the space of possible trajectories becomes more constrained by the observation.

Under assumption (2), both $P(O|\tau)$ and $P(\tau|\mathcal{M}')$ are constant and can be taken out of the equation so we have that equation (1) simplifies once again to equation (3) with $\mathcal{M}^*$ being any model in M^*.

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 are modeling the editing 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 $P(X = \bot) = 1 - p$. Using a Bernoulli process implies that editions are uniformly random and independent (Devroye, Györfi, and Lugosi 2013).

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:

$$P(\mathcal{M}'|\mathcal{M}) = p^d (1-p)^{N-d} \tag{4}$$

where $d = \delta(\mathcal{M}, \mathcal{M}')$, and $p < 0.5$ since we consider that the cost of applying an *edit operation* is higher than not applying it. This means that the lower the value of d, the closer a model M' is to the original model M and so the more likely it is. Note also that all models at a same distance d will be assigned the same $P(\mathcal{M}'|\mathcal{M})$ by this equation, which means that this probability only depends on the two given models and is "blind" to the observation.

The $P(\mathcal{M}^*|\mathcal{M})$ probability is then given by equation (4), and computing the distance from $\mathcal{M}$ to the *closest consistents models* $\mathcal{M}^*$ is given by the *observation distance*, $d = \delta^o(\mathcal{M}, \mathcal{O})$.

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 $\mathcal{O}(\tau)$.

Essentially, the task $\langle \mathcal{M}, \mathcal{O} \rangle$ of the AJO approach can be used for 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 $\mathcal{M} = \{\xi_1, \ldots, \xi_n\}$, 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 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:

- 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 and removing a precondition pre_e_ξ, a positive effect add_e_ξ or a negative effect del_e_ξ in ξ.
  - 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 1 except that the schema `inc-x` is defined without preconditions and its positive/negative effects are swapped with respect to Figure 1. And consider an observation that only contains the initial and final state of Figure 2; i.e., $\mathcal{O}(\tau) = \langle s_0^o, s_m^o \rangle = \langle \{$`(xcoord 0)(ycoord 0)`$\}, \{$`(xcoord 2)(ycoord 1)`$\} \rangle$. The plan found by a planner to P' is shown in figure 4. The first seven steps

```
00 : (insert_pre_xcoord_v1_inc-x)    07 : (validate_0)
01 : (insert_pre_next_v1_v2_inc-x)   08 : (apply_inc-x 0 1)
02 : (insert_pre_q0_inc-x)           09 : (apply_inc-x 1 2)
03 : (delete_del_xcoord_v2_inc-x)    10 : (apply_inc-x 2 3)
04 : (delete_add_xcoord_v1_inc-x)    11 : (apply_inc-y-even 0 1)
05 : (insert_del_xcoord_v1_inc-x)    12 : (apply_dec-x 3 2)
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 1 and $\mathcal{O}$ a partial observation from Figure 2.

are the edit actions to *fix* the schema `inc-x`; step 07 is a validate action that sets the robot in the initial state $s_0^o = \langle \{$`(xcoord 0)(ycoord 0)`$\} \rangle$; step 08 applies the action `(inc-x 0 1)` and moves the robot one cell to the right to position $(1, 0)$; steps 09-11 also move the robot one cell to the right; step 11 applies an action that increases coordinate y from a position in an even row number, thus moving robot to row 1; 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 $s_m^o = \langle \{$`(xcoord 2)(ycoord 1)`$\} \rangle$.

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

In this section, we evaluate the empirical performance of our approach[3] in three different applications of *model recognition*. We will assume in the three experiments that $\mathcal{O}(\tau)$ contains an *empty sequence of observed actions* and so we will only work with the available observed states.

For each experiment, we define a set M of different planning models that share the same state variables but update the variables using different action schemata. For every $\mathcal{M} \in M$, we generated the same number of partial observations of plan trajectories ($\mathcal{O}(\tau)$) with such a model. Finally, we applied our *model recognition as planning* approach to identify the model, among the models in M, which was actually used for generating a given $\mathcal{O}(\tau)$ observation. We assume equiprobable prior probabilities in all the experiments; i.e., $P(\mathcal{M}) = \frac{1}{|M|}$ for every $\mathcal{M} \in M$.

In order to provide a better explanation of the experiments below, we introduce a particular class of $\mathcal{O}(\tau)$ observations. This new class allows us to distinguish between *observable* state variables, whose value may be read from sensors, and *hidden* or *latent* state variables 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.*

Hence the value of the variables in $\Phi \subseteq F$ is observable while the value of the variables in $F \setminus \Phi$ is unobservable.

Reproducibility. All experiments were run in an Intel Core i5 3.10GHz x 4 16GB of RAM and the classical plan-

[3] Available at https://github.com/anonsub/model-recognition.

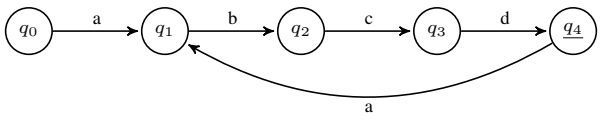

Figure 5: A 4-symbol and 5-state regular automata for recognizing the $(abcd)^+$ language ($\underline{q_4}$ is the acceptor state).

ner 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 deadends (López, Celorrio, and Olaya 2015). Other planners, such as FastDownward were also tested but provided worse experimental results. We set up a timeout of 1000s for the computation of $\delta^o(\mathcal{M}, \mathcal{O})$, at which point it is assigned the maximum distance.

Recognition of *regular automata*

The first experiment, which doubles as a proof of concept, exploits *model recognition as planning* for a classical *string classification* problem. In this experiment, the system receives (1) the string to classify (the $\mathcal{O}(\tau)$ observation) and (2) a set M of different planning models, where each $\mathcal{M} \in M$ represents a different class; i.e. a different regular automata that accepts strings that belong to the class.

Figure 5 illustrates a 4-symbol and 5-state *regular automata* for recognizing the $(abcd)^+$ language. The *input alphabet* is $\Sigma = \{a, b, c, d\}$, and the machine states are $Q = \{q_0, q_1, q_2, q_3, \underline{q_4}\}$, where $\underline{q_4}$ is the only acceptor state. For instance, executing the planning model that encodes the *regular automata* of Figure 5 with the input string $abcdabcd$ produces the following 8-action plan $(\langle a, q_0\rangle \rightarrow q_1)$, $(\langle b, q_1\rangle \rightarrow q_2)$, $(\langle c, q_2\rangle \rightarrow q_3)$, $(\langle d, q_3\rangle \rightarrow \underline{q_4})$, $(\langle a, \underline{q_4}\rangle \rightarrow q_1)$, $(\langle b, q_1\rangle \rightarrow q_2)$, $(\langle c, q_2\rangle \rightarrow \underline{q_3})$, $(\langle d, q_3\rangle \rightarrow \underline{q_4})$.

In this experiment, M comprises five planning models, each representing a different 5-state and 4-symbol regular automata. The five regular languages defined by these 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, $\mathcal{O}(\tau)$ observations, which lengths range from 20 to 30 symbols, thus generating a total of 100 strings. More precisely, $\mathcal{O}(\tau)$ are observations in which the applied transitions (actions) are unobservable and the fluents of the automata internal state are also unobservable. Consequently, $\mathcal{O}(\tau)$ are Φ-observations where the problem states are only represented with the fluents that specify the string to classify.

Table 1 shows the *confusion matrix* resulting from classifying the 100 strings with our method. In this matrix, rows represent the actual class and columns represent the class predicted by our *model recognition as planning* approach.

	$\mathcal{L}1$	$\mathcal{L}2$	$\mathcal{L}3$	$\mathcal{L}4$	$\mathcal{L}5$
$\mathcal{L}1$	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.

Despite using a suboptimal classical planner, we can observe that the class for all the 100 input strings was correctly recognized, which proves the feasibility of our method for classification tasks. The outstanding results reveal that our approach is very suitable for the classification of generative planning models that are more restrictive than STRIPS models, as it is the case of regular automata.

Recognizing failures in a non-deterministic *blocksworld*

With this second experiment we aim to validate the two assumptions underlying our approximation of the $P(\mathcal{O}|\mathcal{M})$ likelihood: (1) that agents adjust their model rationally so $P(\mathcal{O}|\mathcal{M})$ is dominated by the closest consistent model that explains the observation $\mathcal{O}(\tau)$ and (2) that this model can only produce a single trajectory consistent with $\mathcal{O}(\tau)$. Our assumptions are more correct with higher observability and become tautological when the trajectory is fully observed ($\mathcal{O}(\tau) = \tau$).

The input observations $\mathcal{O}(\tau)$ come from a non-deterministic *blocksworld* in which some of the executed actions may have failed. We considered failures of three kinds:

- the execution of a `stack` action fails and causes no effect
- the execution of `unstack` fails and causes no effect
- `unstack` fails and drops the block on the table

M comprises three different 4-schema *blocksworld* models, each one extended with an additional action schema that encodes one of the three above failures. This means that failures are identified by finding the model that better explains the input observation.

The validation of the assumptions is done by measuring the *accuracy* of our approach across different degrees of observability. The accuracy metric is defined as the number of correct predictions over all predictions made, and the degree of observability indicates the probability of observing a fluent in an intermediate state of $\mathcal{O}(\tau)$. In this experiment, we also assume that the length of the trajectory is known. More precisely, at least one fluent of every state of $\mathcal{O}(\tau)$ is observed and the length of the plan trajectory is thus fixed by the observation. This allows us to produce a controlled experiment in which the number of possible trajectories able to explain an observation is bounded.

Figure 6 shows the *accuracy* of our approach when identifying failures over 30 different observations across different degrees of observability. The positive trend seen in the figure proves that the more complete the observations are, the more accurate our estimation becomes.

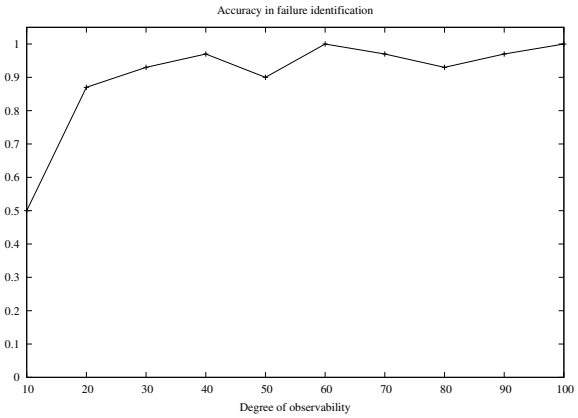

Figure 6: Accuracy for the recognition of failures in a non-deterministic blocksworld.

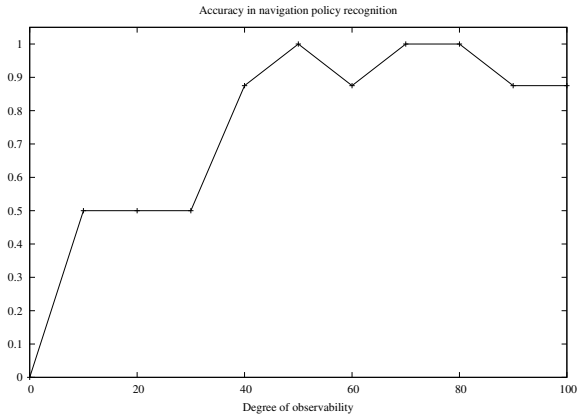

Figure 7: Classification accuracy for the recognition of navigation models.

Recognition of navigation policies

The novel aspect of this experiment is that, unlike the previous experiment, we do not guarantee that the observed states contain at least one fluent. This means that some intermediate states of $\mathcal{O}(\tau)$ may be missing (no fluents are observed) and, consequently, more than a single action may be required to reach an observed state from another one.

The planning models of M represent different navigation models for $N \times N$ grids. In particular, we adopt a *classical navigation model* with actions `up`, `down`, `right`, `left` (to move one cell in each of these four directions) and extend it to define 8 different navigation policies with respect to these additional state variables {`(q0)`, `(q1)`, `(min ?v)`, `(max ?v)`}. An example of a navigation policy is shown in Figure 1. This policy allows to move right when q_0 holds while it allows to move left when q_1 holds, producing a zigzag pattern when visiting all the cells of $N \times N$ grids.

Another interesting aspect of this experiment is that all navigation policies are at a maximum distance of four editions from the base *classical navigation model* that can move the robot in any direction at any given state. Thus, for any given model and observation, a maximum of four editions are needed for the input model to explain the observation. This aspect heavily constrains the discriminating power of $P(\mathcal{O}|\mathcal{M})$, which along with the low degree of observability of the states, gives rise to a meaningful benchmark for *model recognition as planning*.

On the other hand, we generated observations from 8 different trajectories, one for each of the 8 previous planning models. The trajectories depict paths followed by a navigation policy 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, only the values of the fluents encoding the *x* and *y coordinates* of the agent are known; i.e. Φ comprises all the fluents instantiated with predicates `(xcoord ?v)` and `(ycoord ?v)`.

Figure 7 shows the classification *accuracy* achieved by our approach with respect to a range of degrees of observability, from 0% to 100% 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 navigation policies, but from 10% of observability onwards, we start to correctly classify half of the observations. Accuracy stabilizes after 40% observability in the range 0.875 to 1 which means that at most only one out of the 8 observations was not correctly classified.

Conclusions and Related work

This paper formalizes the task of *model recognition* and introduces a novel method that estimates the probability of a STRIPS model to explain a partial observation of a plan execution. Our *model recognition as planning* approach builds on top of off-the-shelf classical planning algorithms and it is robust to missing actions and incomplete or missing intermediate states in the observation. Once the planning model of the observed agent is recognized, the model-based machinery for automated planning becomes fully applicable for other recognition tasks like *goal recognition* (Ramírez and Geffner 2010; Ramírez 2012), *goal recognition design* (Keren, Gal, and Karpas 2014) or *counter-planning* (Pozanco et al. 2018), which require beforehand a model of the observed agent.

We show the effectiveness of our approach in three experiments that encode a set of different models. The first experiment is a proof-of-concept *classification task* that recognizes the regular automata of a given string. The second experiment, where the number of trajectories explaining the observation is bounded, shows that a more accurate model prediction is achieved with more complete observations. The robustness of the approach is shown in the third experiment where actions and intermediates states are unobservable. A significant lesson from the experiments is that our approach allows us to predict behaviours that are not ruled by a goal-driven deliberative planning model; e.g., models of fixed programs or finite state machines (regular automata).

Acknowledgments

This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R. D. Aineto is partially supported by the *FPU16/03184* and S. Jiménez by the *RYC15/18009*. M. Ramírez research is partially funded by DST Group Joint & Operations Analysis Division.

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.

Amir, E., and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research* 33:349–402.

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)*.

Bryce, D.; Benton, J.; and Boldt, M. W. 2016. Maintaining evolving domain models. In *International Joint Conference on Artificial Intelligence, (IJCAI-2016)*, 3053–3059.

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

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.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2018. Human-aware planning revisited: A tale of three models. In *IJCAI-ECAI XAI/ICAPS XAIP Workshops*.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review* 28(02):195–213.

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.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *International Conference on Automated Planning and Scheduling, (ICAPS-14)*, 154–162.

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.

Pozanco, A.; E.-Martín, Y.; Fernández, S.; and Borrajo, D. 2018. Counterplanning using goal recognition and landmarks. In *International Joint Conference on Artificial Intelligence, (IJCAI-18)*, 4808–4814.

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., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *National Conference on Artificial Intelligence, (AAAI-10)*.

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.; Celorrio, S. J.; and Jonsson, A. 2019. Computing programs for generalized planning using a classical planner. *Artificial Intelligence*.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2018. Computing hierarchical finite state controllers with classical planning. *Journal of Artificial Intelligence Research* 62:755–797.

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.

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2018. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, 518–526.

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

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.