Model Recognition as Planning

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

Abstract

Given the partial observation of a plan execution and a set of possible planning models (models that share the same state variables but that update these variables according to different action models), *model recognition* is the task of identifying which model in the set explains/produced the given 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, that we called *model recognition as planning*, is built on top of off-the-shelf classical planning algorithms and is robust to missing intermediate states and actions in the observed plan execution. The effectiveness of *model recognition as planning* is shown in a set of STRIPS models encoding different kinds of *automata*. We show that *model recognition as planning* succeeds to identify the executed automata despite the internal machine state or actual applied transitions, 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 (Ramírez 2012; Ramírez and Geffner 2009). This approach assumes that the action model of the observed agent is known and leverages it to compute the most likely goal of the agent, according to the observed plan execution.

This paper formalizes the *model recognition* task where the object to recognize is not a goal but the *planning model* that determines the behavior of the observed agent. Given a partial observation of a 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.

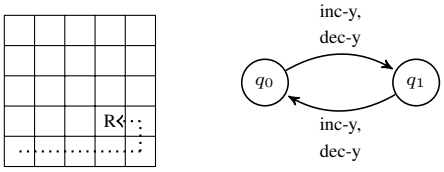

Figure 1: *(Left)* Robot navigating a 5×5 grid. *(Right)* Automata controling that the robot only increments its x-coordinate at *even* rows (when q_0 holds, given that actions `inc-y` and `dec-y` update the robot y-coordinate and the automata state).

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 could increment its *x-coordinate* at *even rows* (when $q0$ holds) and decrement it at the *odd rows* (when $q1$ holds). Apart from this particular navigation model, different action models could be defined within the same state variables (e.g. altering the way $q0$ and $q1$ are required and updated) and these models can determine different kinds of robot navigation. Given an observation of a plan execution (like the one illustrated at Figure 1) *model recognition* would aim here to identify which navigation model produced/explains that observation, despite key information is unobserved (e.g. the particular applied actions or the value of the state variables $q0$ and $q1$).

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 automatae representations, like *finite state controllers*, *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; Ivankovic and Haslum 2015; Segovia-Aguas, Jiménez, and Jonsson 2017).

```
(: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 (?y1 ?y2)
  :precondition (and (ycoord ?y1) (next ?y1 ?y2) (q0))
  :effect (and (not (ycoord ?y1)) (ycoord ?y2)
               (not (q0)) (q1))

(:action inc-y-odd
  :parameters (?y1 ?y2)
  :precondition (and (ycoord ?y1) (next ?y1 ?y2) (q1)))
  :effect (and (not (ycoord ?y1)) (ycoord ?y2)
               (not (q1)) (q0)))

(:action dec-y-even
  :parameters (?y1 ?y2)
  :precondition (and (ycoord ?y1) (next ?y2 ?y1) (q0))
  :effect (and (not (ycoord ?y1)) (ycoord ?y2)
               (not (q0)) (q1)))

(:action dec-y-odd
  :parameters (?y1 ?y2)
  :precondition (and (ycoord ?y1) (next ?y2 ?y1) (q1))
  :effect (and (not (ycoord ?y1)) (ycoord ?y2)
               (not (q1)) (q0)))
```

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

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 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 (same input *alphabet* and same *machine states*) but different *transition functions*. We show that *model recognition as planning* succeeds to identify the executed *automata* despite internal machine states or actual applied transitions are unobserved.

Background

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

Classical planning

We use F to denote the set of *fluents* (propositional variables) describing a state. A *literal* l is a valuation of a fluent $f \in F$, i.e. either $l = f$ or $l = \neg f$. A set of literals L represents a partial assignment of values to fluents (without loss of generality, we will assume that L does not assign conflicting values to any fluent). We use $\mathcal{L}(F)$ to denote the set of all literal sets on F, i.e. all partial assignments of values to fluents.

A *state* s is a full assignment of values to fluents and we explicitly include negative literals $\neg f$ in states; i.e. $|s| = |F|$, so the size of the state space is $2^{|F|}$. Like in PDDL (Fox and Long 2003), we assume that fluents F are instantiated from a set of *predicates* Ψ. Each predicate $p \in \Psi$ has an argument list of arity $ar(p)$. Given a set of *objects* Ω, the set of fluents F is induced by assigning objects in Ω to the arguments of predicates in Ψ; i.e. $F = \{p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}\}$ such that Ω^k is the k-th Cartesian power of Ω.

A *classical planning frame* is a $\langle F, A\rangle$ pair, where F is a set of *fluents* and A is a set of *actions*. The semantics of actions $a \in A$ are specified with two functions: $\rho(s, a)$ that denotes whether the action is *applicable* in a state s and $\theta(s, a)$ that denotes the *successor state* that results of applying the action a in a state s. In this work we specify action semantics (the ρ and θ functions) with the STRIPS model. With this regard, an action $a \in A$ is defined by:

- $\mathsf{pre}(a) \in \mathcal{L}(F)$, the *preconditions* of a, is the set of literals that must hold for the action $a \in A$ to be applicable.
- $\mathsf{eff}^+(a) \in \mathcal{L}(F)$, the *positive effects* of a, is the set of literals that are true after the application of action $a \in A$.
- $\mathsf{eff}^-(a) \in \mathcal{L}(F)$, the *negative effects* of a, is the set of literals that are false after the application of the action.

We say that an action $a \in A$ is *applicable* in a state s iff $\mathsf{pre}(a) \subseteq s$. The result of applying a in s is the *successor state* denoted by $\theta(s, a) = \{s \setminus \mathsf{eff}^-(a)) \cup \mathsf{eff}^+(a)\}$.

A *classical planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where I is an initial state and $G \in \mathcal{L}(F)$ is a goal condition. A *plan* π for P is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$ that induces the *trajectory* $\tau(\pi, s_0) = \langle a_1, s_1, \ldots, a_n, s_n\rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, a_i is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$. The *plan length* is denoted with $|\pi| = n$. A plan π *solves* P iff $G \subseteq s_n$, i.e., if the goal condition is satisfied at the last state reached after following the application of the plan π in the initial state I. A solution plan for P is *optimal* if it has minimum length.

Conditional effects

Conditional effects allow planning actions to have different semantics according to the value of the current state. This feature is useful for compactly defining our method for *model recognition as planning*.

An action $a \in A$ with conditional effects is defined as a set of *preconditions* $\mathsf{pre}(a) \in \mathcal{L}(F)$ and a set of *conditional effects* $\mathsf{cond}(a)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a)$ is composed of two sets of literals $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*.

An action $a \in A$ is *applicable* in a state s iff $\mathsf{pre}(a) \subseteq s$, and the *triggered effects* resulting from the action application are the effects whose conditions hold in s:

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

The result of applying action a in state s is the *successor* state $\theta(s, a) = \{s \setminus \mathsf{eff}_c^-(s, a)) \cup \mathsf{eff}_c^+(s, a)\}$ 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 classical planning problem $P = \langle F, A, I, G\rangle$ and a plan π that solves P; the observation of the execution of π on P is $\mathcal{O}(\pi, I) = \langle a_1^o, s_1^o \ldots, a_l^o, s_m^o\rangle$, an interleaved combination of $1 \leq l \leq |\pi|$ observed actions and $1 \leq m \leq |\pi|$ observed states such that:

- Observed actions are *consistent* with π (Ramírez and Geffner 2009). This means that $\langle a_1^o, \ldots, a_l^o\rangle$ is a subsequence of the solution plan π.
- Observed states are a sub-sequence of partial states *consistent* with the sequence of states $\langle s_0, s_1, \ldots, s_n\rangle$ traversed by π.

On the one hand, the initial state I is fully observed while the observed states in $\mathcal{O}(\pi, I)$ may be partial, i.e. the value of certain fluents in the intermediate states may be omitted ($|s_i^o| \leq |F|$ for every $1 \leq i \leq m$). On the other hand, the sequence of observed states $\langle s_1^o, \ldots, s_m^o\rangle$ in $\mathcal{O}(\pi, I)$ is the same sequence of states traversed by π but certain states may also be omitted. Formally, $0 \leq |s_i^o|$ for every $1 \leq i \leq m$. This means that the transitions between two consecutive observed states in $\mathcal{O}(\pi, I)$ 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 and unbound). Therefore we can conclude that having $\mathcal{O}(\pi, I)$ does not implies knowing the actual length of π.

Definition 1 (Φ-observation)**.** *Given a subset of fluents $\Phi \subseteq F$ we say that $\mathcal{O}(\pi, I)$ 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.*

Figure 1 illustrates the six-state Φ-observation {`<(xcoord 2)(ycoord 1)>`, `<(xcoord 3)(ycoord 1)>`, `<(xcoord 4)(ycoord 1)>`, `<(xcoord 5)(ycoord 1)>`, `<(xcoord 5)(ycoord 2)>`, `<(xcoord 4)(ycoord 2)>`}, where Φ only contains fluents of the kind `(xcoord ?v)` and `(ycoord ?v)`. This means that, for each observed state, only the value of fluents `(xcoord ?v)` and `(ycoord ?v)` is known while the value of the remaining fluents, namely `(next ?v1 ?v2)`, `(q0)` and `(q1)`, is unknown.

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 *classical planning problem* s.t. the semantics of actions $a \in A[\cdot]$ is unknown, i.e. the corresponding functions $\rho(s, a)$ and/or $\theta(s, a)$ are undefined.
- $M = \{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$ is a finite non-empty *set of models* for the actions in A s.t., each model in $\mathcal{M} \in M$, defines different function pairs $\langle \rho, \theta\rangle$ over the state variables F.
- $\mathcal{O}(\pi, I)$ 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 with a different planning model $\mathcal{M} \in M$ and the observed plan execution $\mathcal{O}(\pi, I)$ is the single example to classify. Further, the planning model associated to each class acts as the corresponding *class prototype* that summarizes all the plan executions that could be synthesized with that model (i.e. all the examples that belong to that class).

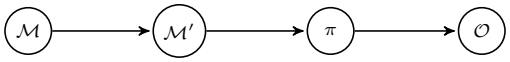

Figure 3: *Bayesian network* abstracting that model $\mathcal{M}'$ results from editing $\mathcal{M}$ and that can produce a plan π consistent with $\mathcal{O}(\pi, I)$.

In this work we follow the *naive Bayes classifier* to assign a model $\mathcal{M} \in M$ to the given observation $\mathcal{O}(\pi, I)$ with respect to the expression:

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

The hypotheses in *model recognition* are then about the possible action models $\mathcal{M} \in M$, while the observation $\mathcal{O}(\pi, I)$ of a plan execution represents the input observation. The *solution* to the *model recognition* task is the model $\mathcal{M} \in M$ (or subset of models in M) that maximizes the previous expression.

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 in our formulation for *model recognition* is the definition of the $P(\mathcal{O}|\mathcal{M})$ *likelihood* that expresses the probability of observing $\mathcal{O}(\pi, I)$ when $\mathcal{M}$ is the planning model.

Our approach to formulate the $P(\mathcal{O}|\mathcal{M})$ *likelihood* is to assess how far is $\mathcal{M}$ from a model $\mathcal{M}'$ that can produce a plan π that solves P and s.t. $\mathcal{O}(\pi, I)$ is a *consistent* observation of the execution of π on P. Figure 3 shows the four-variable *Bayesian network* abstracting this process. Regarding this network we have that:

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

where $\mathcal{M}'$ ranges over all the models that can be generated editing $\mathcal{M}$ and π ranges over all the plans that can be synthesized with a model $\mathcal{M}'$

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

Following the previous equation (2) the exact computation of $P(\mathcal{O}|\mathcal{M})$ is intractable because, for most planning problems, the set of *valid* plans consistent with an observation can easily be very large (infinite in the case of planning problems without dead-ends). Instead, in this work we propose to estimate $P(\mathcal{O}|\mathcal{M})$ making the following assumptions:

1. The sum in the previous equation is dominated by the *closest compliant set*, i.e. the models closest to $\mathcal{M}$ that can produce a solution plan consistent with $\mathcal{O}(\pi, I)$. The rationale behind this assumption is that the further the compliant model, the lower the $P(\mathcal{O}|\mathcal{M})$ likelihood.

The *full observability of the executed plan* is a too strong assumption but, as we explain next, it allows us to build reasonable estimates of the $P(\mathcal{O}|\mathcal{M})$ *likelihood* for the more general case where the observability of the executed plan is partial.

Note that in this baseline scenario where there is *full observability of the executed plan*:

- There is only a single possible plan *consistent* with the given observation, besides $P(\mathcal{O}|\pi) = 1$.
- If we consider assumption 1, the *closest compliant set* $\mathcal{M}^*$ becomes a singleton for this particular scenario, so that probabilities corresponding to different compliant models are not added up. Further, the $P(\pi|\mathcal{M}^*)$ probability equals also to 1 since there is only one possible plan consistent with the observation and by definition, the *closest compliant* model produces that plan.

As a consequence, when there is *full obsevability of the executed plan* and under the previous assumption, we have that expression (1) simplifies to

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

Next we show how to approximate the $P(\mathcal{O}|\mathcal{M})$ *likelihood* for the more general case where there is partial observability of the executed plan. First we start with the simpler scenario where the length of the observed plan is bound and add these two new assumptions.

2. The *closest compliant set* is a singleton, so that probabilities corresponding to different compliant models are not added up.
3. All solutions of a given *plan length* are equiprobable, with shorter plans more likely than longer plans.

In such an scenario $P(\mathcal{O}|\pi)$ equals to a constant α that indicates the number of plans of bounded length that are consistent with $\mathcal{O}$. Likewise, $P(\pi|\mathcal{M}^*)$ is also one given the definition of the *closest compliant set*. As a consequence, when there is *partial obsevability of the executed plan* (but the plan length is bound), and under the previous three assumptions, we have that expression (1) simplifies to

$$argmax_{\mathcal{M}\in M} P(\mathcal{M}^*|\mathcal{M}) \times \alpha. \quad (4)$$

where α is a normalizing constant that has the same value for every $\mathcal{M} \in M$, so it can be ignored.

Finally, we can say that the $P(\mathcal{O}|\mathcal{M})$ *likelihood* can be estimated for the general case of partial observability of the executed plan (where the length of the plan is not bound) by adding a fourth assumption:

4. The observed plan π that solves P is an optimal plan since we are assuming that the observed agent is acting rationally (Ramírez 2012).

In this case the length of the plan becomes bound by the optimal plan length and hence, we can again use expression (4) for the likelihood estimation.

Recognition of STRIPS models

Here we analyze the particular instantiation of the *model recognition* task where the semantics of actions (i.e. ρ and θ functions) are specified with STRIPS action schemas. We start formalizing the STRIPS schema and define the full space of possible STRIPS schema. Eventually, we introduce an *edit distance* for STRIPS schema to estimate the $P(\mathcal{O}|\mathcal{M})$ likelihoods for classical planning models.

Well-defined STRIPS action schema

STRIPS action schema provide a compact representation for specifying classical planning models. For instance, Figure 2 shows six STRIPS action schema, codded in PDDL, that determine a particular kind of robot navigation in $n \times n$ grids.

A STRIPS action schema ξ is defined by 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.

We say that two STRIPS schemes ξ and ξ' are *comparable* iff $pars(\xi) = pars(\xi')$, both share the same list of parameters. For instance, we claim that the six action schema of Figure 2 are *comparable* while, for example, the `stack(?v1,?v2)` and `pickup(?v1)` schemes from the four opertator *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 schema $\xi' \in \mathcal{M}'$ and vice versa.

Given a STRIPS action schema ξ, let us define an additional set of objects ($\Omega \cap \Omega_\xi = \emptyset$), that we denote as *variable names*, and that contains one variable name for each parameter in $pars(\xi)$, that is $\Omega_\xi = \{v_i\}_{i=1}^{|pars(\xi)|}$. For any of the six schema defined in Figure 2, $|pars(\xi)| = 2$ so $\Omega_\xi = \{v_1, v_2\}$.

Given a STRIPS action schema ξ and a set of *predicates* Ψ that shape the propositional state variables. The set of FOL interpretations of Ψ over the corresponding Ω_ξ objects (the *variable names* for schema ξ), confines the elements that can appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ lists. We denote this set of FOL interpretations as $\mathcal{I}_{\Psi,\xi}$. For any of the six schema defined in 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 from $\mathcal{I}_{\Psi,\xi}$ can *a priori* appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ lists of a schema ξ. The space of possible STRIPS schema is bound further by $\mathcal{C}$, a set of constraints of the following kinds:

- *Syntactic constraints*. STRIPS constraints require negative effects appearing as preconditions, negative effects cannot be positive effects at the same time and also, positive effects cannot appear as preconditions. Formally, $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$). Considering exclusively these syntactic constraints, the size of the space of possible STRIPS schema is given by the expression, $2^{2\times|\mathcal{I}_{\Psi,\xi}|}$. For the navigation model of Figure 2, $2^{2\times 10} = 1,048,576$ for every action schema.
- *Domain-specific constraints*. One can also introduce domain-specific knowledge to more precisely bound the space of possible STRIPS schema for a particular domain. For instance, in a *robot navigation* model, like the one in Figure 2, `q0()` and `q1()` are exclusive so they cannot hold a 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 at any of these lists because the `next` predicate is coding

the *succesor* function for *natural numbers*. In this case, introducing these domain-specific constraints reduces the size of the space of possible schema to $2^{2\times7} = 16,384$ for every action schema.

Now we are ready to define what is a *well-defined* STRIPS action schema.

Definition 2 (Well-defined STRIPS action schema)**.** *Given a set of* predicates Ψ*, a list of action* parameters $pars(\xi)$*, and set of FOL constraints* $\mathcal{C}$ *we say that* 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 that an action model is *well-defined* if all its STRIPS action schema are *well-defined*.

Edit distances for STRIPS action models

Edit distances are similarity metrics, traditionally computed over *strings* or *graphs*, and that has been proved successful for *pattern recognition* (Masek and Paterson 1980; Bunke 1997). In this work we formalize and compute edit distances that are referred to STRIPS planning models. First, we define two edit *operations* on a schema $\xi \in \mathcal{M}$ that belongs to a STRIPS action model $\mathcal{M} \in M$:

- *Deletion*. An element is removed from any of these three lists $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of the operator schema $\xi \in \mathcal{M}$ such that the resulting schema is a *well-defined* STRIPS action schema.
- *Insertion*. An element in $\mathcal{I}_{\Psi,\xi}$ is added to any of these three lists $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of the operator schema $\xi \in \mathcal{M}$ s.t. the resulting schema is *well-defined*.

We can now formalize an *edit distance* that quantifies how similar two given STRIPS action 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 *action models defined 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 $\mathcal{I}_{\Psi,\xi}$ are bound sets, the maximum number of edits that can be introduced to a given action model is bound as well.

Definition 4 (Maximum edit distance)**.** *The* ***maximum edit distance*** *of an* STRIPS *model* $\mathcal{M}$ *built within the set of predicates* Ψ *is* $\delta(\mathcal{M}, *) = \sum_{\xi\in\mathcal{M}} 3 \times |\mathcal{I}_{\Psi,\xi}|$*.*

The observation of a plan execution, generated with an action model $\mathcal{M}$, reflects *semantic knowledge* that constrain further the space of the possible schema $\xi \in \mathcal{M}$. In this case, we talk about *observation constraints* (that can also be included into the $\mathcal{C}$ set). In addition, *observation constraints* allow us to define an edit distance to asses the matching of a STRIPS model with respect to an observation of a plan execution.

```
00 : (insert_pre_xcoord_v1_inc-x)    07 : (validate_0)
01 : (insert_pre_next_v1_v2_inc-x)   08 : (editable_inc-x 1 2)
02 : (insert_pre_q0_inc-x)           09 : (editable_inc-x 2 3)
03 : (delete_del_xcoord_v2_inc-x)    10 : (editable_inc-x 3 4)
04 : (delete_add_xcoord_v1_inc-x)    11 : (editable_inc-x 4 5)
05 : (insert_del_xcoord_v1_inc-x)    12 : (editable_inc-y-even 1 2)
06 : (insert_add_xcoord_v2_inc-x)    13 : (editable_dec-x 5 4)
                                     14 : (validate_1)
```

Figure 4: Plan for editing (steps [0-6]) and validating (steps [7-14]) the planning model of Figure2 when action `inc-x` has no preconditions and positive/negative effects are swapped wrt Figure2.

Definition 5 (Observation edit distance)**.** *Given* $\mathcal{O}(\pi, I)$*, an observation of the execution of a plan for solving* P *and a* STRIPS *action model* $\mathcal{M}$*, all defined within the same set of predicates* Ψ*. The* **observation edit distance***,* $\delta(\mathcal{M}, \mathcal{O})$*, is the minimal edit distance from* $\mathcal{M}$ *to any* comparable *and well-defined model* $\mathcal{M}'$ *s.t.* $\mathcal{M}'$ *produces a plan* π *that solves* P *and that is* consistent *with* $\mathcal{O}(\pi, I)$*;*

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

Remarkably, the *observation edit distance* allows us to elicit the likelihood of the observations given a candidate model. It can be argued that the shorter this distance the better the given model explains the given observation. Note that the *observation edit distance* could also be defined assessing the edition required by the observed plan execution to match the given model. This implies defining *edit operations* that modify the observation $\mathcal{O}(\pi, I)$ instead of $\mathcal{M}$ (Sohrabi, Riabov, and Udrea 2016). Our definition of the *observation edit distance* is more practical since normally $\mathcal{I}_{\Psi,\xi}$ is much smaller than F. In practice, the number of *variable objects* should be smaller than the number of objects in a planning problem.

Definition 6 (*Closest compliant models)***.** *Given a model* $\mathcal{M}$*. The* **closest compliant models***,* $\mathcal{M}^*$*, is the comparable set of action models closest to* $\mathcal{M}$ *(in terms of editions) that can produce a solution plan consistent with* $\mathcal{O}(\pi, I)$*;*

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

Model Recognition as classical planning

This section shows that, for STRIPS planning models, $\delta(\mathcal{M}, \mathcal{O})$ can be computed (and hence an approximation of the $P(\mathcal{O}|\mathcal{M})$ likelihood) with a compilation of a classical planning with conditional effects.

The compilation is an extension of the classical planning compilation for the learning of STRIPS planning models (Aineto, Jiménez, and Onaindia 2018). The intuition behind this compilation is that a solution to the resulting classical planning task is a sequence of actions that:

1. **Edits the action model** $\mathcal{M}$ **to build** $\mathcal{M}'$. A solution plan starts with a *prefix* that modifies the preconditions and effects of the action schemes in $\mathcal{M}$ using to the two *edit operations* defined above, *deletion* and *insertion*.
2. **Validates the edited model** $\mathcal{M}'$. The solution plan continues with a postfix that:

```
;;; Propositional encoding for inc-x(?v1 ?v2)
(pre_xcoord_v1_inc-x) (pre_next_v1_v2_inc-x)
(pre_q0__inc-x)
(del_xcoord_v1_inc-x) (add_xcoord_v2_inc-x)

;;; Propositional encoding for dec-x(?v1 ?v2)
(pre_xcoord_v1_dec-x) (pre_next_v2_v1_dec-x)
(pre_q1___dec-x)
(del_xcoord_v1_dec-x) (add_xcoord_v2_dec-x)

;;; Propositional encoding for inc-y-even(?v1 ?v2)
(pre_ycoord_v1_inc-y-even) (pre_next_v1_v2_inc-y-even)
(pre_q0__inc-y-even)
(del_ycoord_v1_inc-y-even) (del_q0__inc-y-even)
(add_ycoord_v2_inc-y-even) (add_q1__inc-y-even)

;;; Propositional encoding for inc-y-odd(?v1 ?v2)
(pre_ycoord_v1_inc-y-odd) (pre_next_v1_v2_inc-y-odd)
(pre_q0__inc-y-odd)
(del_ycoord_v1_inc-y-odd) (del_q1__inc-y-odd)
(add_ycoord_v2_inc-y-odd) (add_q0__inc-y-odd)

;;; Propositional encoding for dec-y-even(v1 ?v2)
(pre_ycoord_v1_dec-y-even) (pre_next_v2_v1_dec-y-even)
(pre_q0__dec-y-even)
(del_ycoord_v1_dec-y-even) (del_q0__dec-y-even)
(add_ycoord_v2_dec-y-even) (add_q1__dec-y-even)

;;; Propositional encoding for dec-y-odd(?v1 ?v2)
(pre_ycoord_v1_dec-y-odd) (pre_next_v2_v1_dec-y-odd)
(pre_q1__dec-y-odd)
(del_ycoord_v1_dec-y-odd) (del_q1__dec-y-odd)
(add_ycoord_v2_dec-y-odd) (add_q0__dec-y-odd)
```

Figure 5: Propositional encoding for the six schema from Figure 2.

(a) Induces a solution plan π for the original classical planning problem P.

(b) Validates that $\mathcal{O}(\pi, I)$ is an observation of the execution of π on the classical planning problem P.

Figure 4 shows the plan with a prefix (*steps [0,6]*) for editing the planning model of Figure2 when its schema `inc-x` is defined without preconditions and its positive/negative effects are swapped wrt Figure 2. The postfix of the plan (*steps [7,14]*) validates the edited action model at the observation of the plan execution illustrated at Figure 1.

Note that our interest is not in $\mathcal{M}'$, the edited model resulting from the compilation, but in the number of required *edit operations* (insertions and deletions) required by $\mathcal{M}'$ to be validated. In the example of Figure 4 $\delta(\mathcal{M}, \mathcal{O}) = 7$ and $\delta(\mathcal{M}, *) = 6 \times 3 \times 10$ since there are 6 action schemes for which $|\mathcal{I}_{\Psi,\xi}| = 10$.

A propositional encoding for STRIPS action schema

Given a STRIPS action schema ξ, a propositional encoding for the *preconditions*, *negative* and *positive* effects of that schema can be represented with the fluents of the kind $[pre|del|add]\_e\_\xi$ such that $e \in \mathcal{I}_{\Psi,\xi}$ is a single element from the set of interpretations of predicates Ψ over the corresponding objects Ω_ξ. Figure 5 shows the propositional encoding for the six action schema defined in Figure 2.

The interest of having a propositional encoding for STRIPS action schema is that, using *conditional effects*, it allows to compactly define *editable actions*. Actions whose semantics is given by the value of the $[pre|del|add]\_e\_\xi$ fluents on the current state. Given an operator schema $\xi \in \mathcal{M}$

```
(:action editable_inc-x
  :parameters (?v1 ?v2)
  :precondition
    (and (or (not (pre_xcoord_v1_inc-x)) (xcoord ?v1))
         (or (not (pre_xcoord_v2_inc-x)) (xcoord ?v2))
         (or (not (pre_ycoord_v1_inc-x)) (xcoord ?v1))
         (or (not (pre_ycoord_v2_inc-x)) (xcoord ?v2))
         (or (not (pre_q0__inc-x)) (q0))
         (or (not (pre_q1__inc-x)) (q1)))
         (or (not (pre_next_v1_v1_inc-x)) (next ?v1 ?v1)))
         (or (not (pre_next_v1_v2_inc-x)) (next ?v1 ?v2)))
         (or (not (pre_next_v2_v1_inc-x)) (next ?v2 ?v1)))
         (or (not (pre_next_v2_v2_inc-x)) (next ?v2 ?v2))))
    :effect (and
       (when (del_xcoord_v1_inc-x) (not (xcoord ?v1)))
       (when (del_xcoord_v2_inc-x) (not (xcoord ?v2)))
       (when (del_ycoord_v1_inc-x) (not (xcoord ?v1)))
       (when (del_ycoord_v2_inc-x) (not (xcoord ?v2)))
       (when (del_q0__inc-x) (not (q0)))
       (when (del_q1__inc-x) (not (q1)))
       (when (del_next_v1_v1_inc-x) (not (next ?v1 ?v1)))
       (when (del_next_v1_v2_inc-x) (not (next ?v1 ?v2)))
       (when (del_next_v2_v1_inc-x) (not (next ?v2 ?v1)))
       (when (del_next_v2_v2_inc-x) (not (next ?v2 ?v2)))

       (when (add_xcoord_v1_inc-x) (xcoord ?v1))
       (when (add_xcoord_v2_inc-x) (xcoord ?v2))
       (when (add_ycoord_v1_inc-x) (xcoord ?v1))
       (when (add_ycoord_v2_inc-x) (xcoord ?v2))
       (when (add_q0__inc-x) (q0))
       (when (add_q1__inc-x) (q1))
       (when (add_next_v1_v1_inc-x) (next ?v1 ?v1))
       (when (add_next_v1_v2_inc-x) (next ?v1 ?v2))
       (when (add_next_v2_v1_inc-x) (next ?v2 ?v1))
       (when (add_next_v2_v2_inc-x) (next ?v2 ?v2)))
```

Figure 6: Editable version of the `inc-x(?v1,?v2)` schema for robot navigation in a $n \times n$ grid.

its *editable* version is formalized as:

$$\begin{aligned}\mathsf{pre}(\mathsf{editable}_\xi) =& \{pre\_e\_\xi \implies e\}_{\forall e \in \mathcal{I}_{\Psi,\xi}} \\ \mathsf{cond}(\mathsf{editable}_\xi) =& \{del\_e\_\xi\} \rhd \{\neg e\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}, \\ & \{add\_e\_\xi\}\} \rhd \{e\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}.\end{aligned}$$

Figure 6 shows the PDDL encoding of the *editable version* of the `inc-x(?v1,?v2)` schema for robot navigation in a $n \times n$ grid (see Figure 2). Note that this editable schema, when the fluents of Figure 5 hold, behaves exactly as defined in Figure 2.

The compilation formalization

Conditional effects allow us to compactly define our compilation for computing $\delta(\mathcal{M}, \mathcal{O})$ and hence, estimate the $P(\mathcal{O}|\mathcal{M})$ likelihood. Given a STRIPS model $\mathcal{M} \in M$ and the observation $\mathcal{O}(\pi, I)$ of the execution of a plan for solving $P = \langle F, A, I, G\rangle$, our compilation outputs a classical planning task with conditional effects $P' = \langle F', A', I', G'\rangle$ such that:

- F' extends the original fluents F with:
  - Fluents $[pre|del|add]\_e\_\xi$ to model the possible STRIPS schema.
  - The fluents to code the *observation constraints*:
    - $F_\pi = \{plan(name(a_i), \Omega^{ar(a_i)}, i)\}_{1 \le i \le l}$ to code the i^{th} action in $\mathcal{O}(\pi, I)$. The static facts $next_{i,i+1}$ and the fluents at_i, $1 \le i < l$, are also added to iterate through the l actions in $\mathcal{O}(\pi, I)$.

  - $*$ The fluents $\{test_j\}_{1 \leq j \leq m}$, indicating the state observation $s_j \in \mathcal{O}(\pi, I)$ where the action model is validated.
  - The fluents $mode_{edit}$ and $mode_{val}$ to indicate whether the operator schemas are edited or validated.
- I' extends the original initial state I with the fluent $mode_{edit}$ set to true as well as the fluents F_π plus fluents at_1 and $\{next_{i,i+1}\}$, $1 \leq i < l$, for tracking the plan step where the action model is validated. Our compilation assumes that initially $\mathcal{M}'$ is defined as $\mathcal{M}$. Therefore fluents $[pre|del|add]\_e\_\xi$ hold as given by $\mathcal{M}$.
- $G' = G \bigcup \{at_n, test_m\}$.
- A' comprises three kinds of actions with conditional effects:
  1. The *editable* version of the original actions in A. This actions have now an extra preconditions because they can only be applied in the *validation* mode (i.e. when $mode_{val}$ holds). When the observation $\mathcal{O}(\pi, I)$ includes observed actions, they also include the extra conditional effects $\{at_i, plan(name(a_i), \Omega^{ar(a_i)}, i)\} \rhd \{\neg at_i, at_{i+1}\}_{\forall i \in [1,l]}$ to validate that actions are applied, exclusively, in the same order as they appear in $\mathcal{O}(\pi, I)$.

  2. Actions for *editing* operator schema $\xi \in \mathcal{M}$. This includes the actions for *inserting* a new *precondition* into an action schema $\xi \in \mathcal{M}$ and for inserting a new *negative* or *positive* effect into the action schema $\xi \in \mathcal{M}$

$$\begin{aligned} \mathsf{pre}(\mathsf{insertPre}_{\mathsf{e},\xi}) =& \{\neg pre\_e\_\xi, \neg del\_e\_\xi, \\ & \neg add\_e\_\xi, mode_{edit}\}, \\ \mathsf{cond}(\mathsf{insertPre}_{\mathsf{e},\xi}) =& \{\emptyset\} \rhd \{pre\_e\_\xi\}. \end{aligned}$$

$$\begin{aligned} \mathsf{pre}(\mathsf{insertEff}_{\mathsf{e},\xi}) =& \{\neg del\_e\_\xi, \neg add\_e\_\xi, mode_{edit}\}, \\ \mathsf{cond}(\mathsf{insertEff}_{\mathsf{e},\xi}) =& \{pre\_e\_\xi\} \rhd \{del\_e\_\xi\}, \\ & \{\neg pre\_e\_\xi\} \rhd \{add\_e\_\xi\}. \end{aligned}$$

     Besides these actions, A' also contains the actions for *deleting* a *precondition* and a negative/positive *effect*.
  3. Actions for *validating* the edited models at the s_j observed states, $0 \leq j < m$.

$$\begin{aligned} \mathsf{pre}(\mathsf{validate}_{\mathsf{j}}) =& s_j \cup \{test_{j-1}\}, \\ \mathsf{cond}(\mathsf{validate}_{\mathsf{j}}) =& \{\emptyset\} \rhd \{\neg test_{j-1}, test_j, \\ & \{mode_{edit}\} \rhd \{\neg mode_{edit}, mode_{val}\}. \end{aligned}$$

Evaluation

To evaluate the empirical performance of *model recognition as planning* we collect a set M of possible STRIPS models, that share the same state variables but define different action models. Then, we randomly choose one of these models $\mathcal{M} \in M$ and use it to produce an observation $\mathcal{O}(\pi, I)$ of a plan execution. Finally, we follow our *model recognition as planning* method to identify the model $\mathcal{M} \in M$ that produced $\mathcal{O}(\pi, I)$. The experiment is repeated for models of different kind and different observability of the given plan execution.

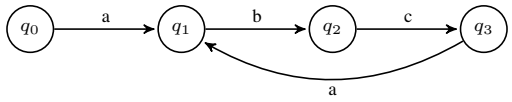

Figure 7: Four-symbol four-state *regular automata* for recognizing the $\{(abc)^n : n \geq 1\}$ language ($\underline{q_3}$ is the acceptor state).

Reproducibility. MADAGASCAR is the classical planner we used to solve the instances that result from our compilations for its ability to deal with dead-ends (Rintanen 2014). Due to its SAT-based nature, MADAGASCAR can apply the actions for editing preconditions in a single planning step (in parallel) because there is no interaction between them. Actions for editing effects can also be applied in a single planning step, thus significantly reducing the planning horizon.

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 Automatae*. In this experiment the models in M represent different *regular automatae*. Figure 7 illustrate a four-symbol four-state *regular automata* for recognizing the $\{(abc)^n : n \geq 1\}$ language. The *input alphabet* is $\Sigma = \{a, b, c, \square\}$, and the machine states are $Q = \{q_0, q_1, q_2, \underline{q_3}\}$ (where $\underline{q_3}$ is the only acceptor state).

In more detail, we randomly generated a $M = \{\mathcal{M}_1, \ldots, \mathcal{M}_{50}\}$ set of fifty different STRIPS models that encode different *regular automata*. Each $\mathcal{M} \in M$ encodes a different five-state *regular automata* with a five-symbol input alphabet. Each automata transition is encoded with a planning action. For instance, the execution of the *regular automatae* defined in Figure 8, with the sequence of input symbols $abcabc$, produces the following six-action plan $(a, q_0 \rightarrow q_1)$, $(b, q_1 \rightarrow q_2)$, $(c, q_2 \rightarrow q_3)$, $(a, q_3 \rightarrow q_1)$, $(b, q_1 \rightarrow q_2)$, $(c, q_2 \rightarrow q_3)$.

Here we assume that the actual applied transitions are unknown as well as the internal machine state. Assuming that the actual applied transitions is unknown means that the observation $\mathcal{O}(\pi, I)$ of the execution of a regular automata contains no actions, it is simply a sequence of states $\mathcal{O}(\pi, I) = \langle s_1, \ldots, s_m \rangle$. Assuming that the internal machine state is unknown means that $\mathcal{O}(\pi, I)$ is a Φ-observation and that the Φ subset does not contain (q) fluents, with $q \in Q$ and $q \neq q_0$.

Recognition of *Turing Machines*. In this experiment the given models in M represent different *Turing machines* (Bylander 1994; Porco, Machado, and Bonet 2013). Figure 8 illustrate a seven-symbol six-state *Turing Machine* for recognizing the $\{a^n b^n c^n : n \geq 1\}$ language. The *tape alphabet* is $\Sigma = \{a, b, c, x, y, z, \square\}$, the *input alphabet* $\Upsilon = \{a, b, c, \square\}$ and the machine states are $Q = \{q_0, q_1, q_2, q_3, q_4, \underline{q_5}\}$ (where $\underline{q_5}$ is the only acceptor state).

Here we randomly generated a $M = \{\mathcal{M}_1, \ldots, \mathcal{M}_{50}\}$ set of fifty different STRIPS models that encode different five-symbol five-state *Turing Machines* with circular tapes.

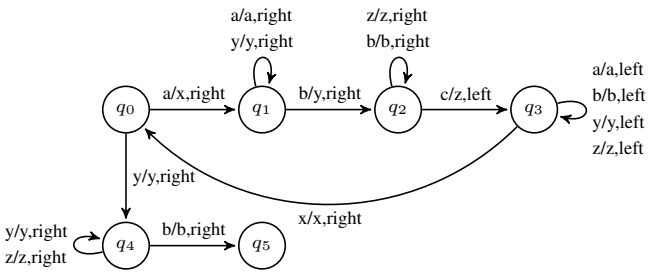

Figure 8: Seven-symbol six-state *Turing Machine* for recognizing the $\{a^n b^n c^n : n \geq 1\}$ language.

There is a planning action $a \in A$ for each machine transition, e.g. the full encoding of the *Turing Machine* of Figure 8 produces a total of sixteen STRIPS action schema. With this regard, the execution of the *Turing Machine* defined in Figure 8 with an initial tape $abc\square\square\square$ produces the eight-action plan $(a, q_0 \rightarrow x, r, q_1)$, $(b, q_1 \rightarrow y, r, q_2)$, $(c, q_2 \rightarrow z, l, q_3)$, $(y, q_3 \rightarrow y, l, q_3)$, $(x, q_3 \rightarrow x, r, q_0)$, $(y, q_0 \rightarrow y, r, q_4)$, $(z, q_4 \rightarrow z, r, q_4)$, $(\square, q_4 \rightarrow \square, r, \underline{q_5})$.

Again we assume that the actual applied transitions is unknown ($\mathcal{O}(\pi, I)$ contains no actions). Further, $\mathcal{O}(\pi, I)$ is a Φ-observation and that the Φ subset does not contain $q()$ fluents, with $q \in Q$ and $q \neq q_0$ and that the values of several tape cells is unknown.

Recognition of navigation models. In this experiment the given models in M represent different navigation models that are computed as the cross product of a regular automata with a four-operator STRIPS model for navigating a $n \times n$ grid.

In this case the regular automata constrain the applications of the navigation actions producing different navigation policies e.g. like the one in Figure 2. Given an observation of a plan execution, like the one illustrated at Figure 1, here the task is to identify which navigation model produced that observation, despite the the applied actions are unobserved. In addition, for each observed state, only the value of fluents encoding the x and y coordinates of the agent are known while the value of the regular automata conditioning the navigation policy is unknown.

Results

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.

In this work we do not assume that the observed agent is acting rationally, like in *plan recognition as planning* (Ramírez 2012; Ramírez and Geffner 2009). 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).

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.

Remarkably, the extension of this piece of work to the FOND planning setting (Muise, McIlraith, and Beck 2012) is straightforward by simply considering the *all-outcomes* determiniztion of the actions with non-determinitic effects (Yoon, Fern, and Givan 2007). An interesting research direction is however to understand how to apply our approach to planning models where the planning models include actions with probabilistic effects (Younes et al. 2005).

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning strips action models with classical planning.

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *ICAPS*, 26–33.

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

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

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence* 69(1-2):165–204.

Carberry, S. 2001. Techniques for plan recognition. *User Modeling and User-Adapted Interaction* 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 *IJCAI*.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Geffner, H., and Bonet, B. 2013. A concise introduction to models and methods for automated planning.

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 *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *ICAPS*.

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

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *ICAPS*.

Porco, A.; Machado, A.; and Bonet, B. 2013. Automatic reductions from ph into strips or how to generate short problems with very long solutions. In *ICAPS*.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *International Joint conference on Artifical Intelligence*, 1778–1783.

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

Rintanen, J. 2014. Madagascar: Scalable planning with sat. *Proceedings of the 8th 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, ICAPS-17*.

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 *IJCAI*, 3258–3264.

Sukthankar, G.; Geib, C.; Bui, H. H.; Pynadath, D.; and Goldman, R. P. 2014. *Plan, activity, and intent recognition: Theory and practice*. Newnes.

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.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, 352–359.

Younes, H. L.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research* 24:851–887.