Planning Model Recognition

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

Abstract

Given a set of possible planning models and an observation of a plan execution, *Planning Model Recognition* is the task of identifying which of these models has the highest probability of producing the given observation. The paper formalizes the *Planning Model Recognition* task and proposes a metric to assess the likelihood of a given STRIPS model to produce the observation of a plan execution. This likelyhood metric is robust to missing intermediate states and actions in the input observation of a plan execution besides it is computable with classical planning. The effectiveness of our method for *Planning Model Recognition* is evaluated in a set of 100 possible STRIPS models each, representing a different *Turing Machine*, all of them sharing the same tape alphabet and same set of machine states.

Introduction

Plan recognition is the task of predicting the future actions of an agent provided the observation of its current behaviour. *Goal recognition* is a subtask of plan recognition with the particular objective of discovering the terminal goal of the observed agent. A large number of methods has been proposed for plan/goal recognition: rule-based systems, parsing (both conventional and stochastic), graph-covering, Bayesian nets, ...

This paper defines *planning model recognition*, the task of identifying the model with the highest probability of producing the observed behaviour of an agent. Related to both *Plan/Goal recognition*, the recognition of planning models is meaningful because:

- It allows to apply the model-based machinery of the crisp framework for *Plan recognition as planning* (Ramírez 2012; Ramírez and Geffner 2009),
- it is useful to determine whether the curent agent behaviour is suboptimal with respect to the identified model. For instance using an optimal planner (Pommerening, Helmert, and Bonet 2017).

The paper introduces a novel metric to assess the likelihood of a given STRIPS model to produce the observation of a plan execution and a method for computing this likelihood using an off-the-shelf classical planner. Remarkabley, this metric is robust to missing intermediate states and actions in the input observation of a plan execution. The effectiveness of our method for *Planning Model Recognition* is evaluated in a set of 100 possible STRIPS models each, representing a different *Turing Machine*, all of them sharing the same tape alphabet and same set of machine states.

Background

This section defines the classical planning model that we follow in this work and STRIPS the action model we use in the *Planning Model Recognition*.

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 contain conflicting values). We use $\mathcal{L}(F)$ to denote the set of all literal sets on F; i.e. all partial assignments of values to fluents.

A *state* s is a full assignment of values to fluents; $|s| = |F|$, so the size of the state space is $2^{|F|}$. Explicitly including negative literals $\neg f$ in states simplifies subsequent definitions but often we will abuse of notation by defining a state s only in terms of the fluents that are true in s, as it is common in STRIPS planning.

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

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.

A *classical planning problem* is a tuple $P = \langle F, A, I, G \rangle$, where I is an initial state and $G \subseteq \mathcal{L}(F)$ is a goal condition. A *plan* for P is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that induces the *state trajectory* $\langle s_0, s_1, \ldots, s_n \rangle$ such that $s_0 = I$ and a_i ($1 \leq i \leq n$) 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 in the last state resulting from the application of the plan π in the initial state I.

STRIPS action schemas

This work addresses the recognition of PDDL action schemas that follow the STRIPS requirement (McDermott et al. 1998; Fox and Long 2003). Figure 1 shows the *stack* action schema, coded in PDDL, from a four-operator *blocksworld* (Slaney and Thiébaux 2001).

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

Figure 1: STRIPS operator schema coding, in PDDL, the *stack* action from a four-operator *blocksworld*.

We assume that fluents F are instantiated from a set of *predicates* Ψ, as in PDDL. 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. $F = \{p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}\}$ s.t. Ω^k is the k-th Cartesian power of Ω.

Let $\Omega_v = \{v_i\}_{i=1}^{\max_{a \in A} ar(a)}$ be a new set of objects ($\Omega \cap \Omega_v = \emptyset$), denoted as *variable names*, and that is bound by the maximum arity of an action in a given planning frame. For instance, in a three-block *blocksworld* $\Omega = \{block_1, block_2, block_3\}$ while $\Omega_v = \{v_1, v_2\}$ because the operators with the maximum arity, `stack` and `unstack`, have arity two. We define F_v, a new set of fluents s.t. $F \cap F_v = \emptyset$, that results from instantiating Ψ using only the objects in Ω_v, i.e. the variable names, and that defines the elements that can appear in an action schema. In *blocksworld* this set contains 11 elements, F_v={`handempty`, `holding`(v_1), `holding`(v_2), `clear`(v_1), `clear`(v_2), `ontable`(v_1), `ontable`(v_2), `on`(v_1,v_1), `on`(v_1,v_2), `on`(v_2,v_1), `on`(v_2,v_2)}.

For a given operator schema ξ, we define $F_v(\xi) \subseteq F_v$ as the subset of fluents that represent the elements that can appear in that action schema. For instance, for the *stack* action schema $F_v(\texttt{stack}) = F_v$ while $F_v(\texttt{pickup})$={`handempty`, `holding`(v_1), `clear`(v_1), `ontable`(v_1), `on`(v_1,v_1)} excludes the fluents from F_v that involve v_2 because the action header `pickup`(v_1) contains the single parameter v_1.

We assume also that actions $a \in A$ are instantiated from STRIPS operator schemas $\xi = \langle head(\xi), pre(\xi), add(\xi), del(\xi)\rangle$ where:

- $head(\xi) = \langle name(\xi), pars(\xi)\rangle$, is the operator *header* defined by its name and the corresponding *variable names*, $pars(\xi) = \{v_i\}_{i=1}^{ar(\xi)}$. The headers of a four-operator *blocksworld* are `pickup`(v_1), `putdown`(v_1), `stack`(v_1,v_2) and `unstack`(v_1,v_2).
- The preconditions $pre(\xi) \subseteq F_v$, the negative effects $del(\xi) \subseteq F_v$, and the positive effects $add(\xi) \subseteq F_v$ such that, $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$.

Given the set of predicates Ψ and the header of the operator schema ξ, $2^{2|F_v(\xi)|}$ defines the size of the space of possible STRIPS models for that operator. Note that the previous constraints require that negative effects appear as preconditions and that they cannot be positive effects and also, that a positive effect cannot appear as a precondition. For instance, $2^{2|F_v(stack)|} = 4194304$ for the blocksworld `stack` operator while for `pickup` is only 1024.

Last but not least, we say that two STRIPS operator schemes ξ and ξ' are *comparable* if both schemas have the same parameters so they share the same space of possible STRIPS models. Formally, if $pars(\xi) = pars(\xi')$ it also holds that $F_v(\xi) = F_v(\xi')$. For instance, we can claim that blocksworld operators `stack` and `unstack` are *comparable* while `stack` and `pickup` are not. Two STRIPS action models $\mathcal{M}$ and $\mathcal{M}'$ are *comparable* iff there exists a bijective function $\mathcal{M} \mapsto \mathcal{M}^*$ that maps every $\xi \in \mathcal{M}$ to a comparable action schema $\xi' \in \mathcal{M}'$ and viceversa.

Planning Model Recognition

Given a set of possible action models $M = \{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$ and an observation of a plan execution $\mathcal{T} = \langle s_0, a_{,1}, s_1, \ldots, a_n, s_n\rangle$ that is obtained watching the execution of a plan $\pi = \langle a_1, \ldots, a_n\rangle$ s.t., for each $1 \leq i \leq n$, a_i is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$. *Planning Model Recognition* is the task of identifying which model $\mathcal{M} \in M$ has the highest probability of producing $\mathcal{T}$.

The STRIPS edit distance

The intuition of our method for *Planning Model Recognition* is to assess how well an action model $\mathcal{M} \in M$ explains $\mathcal{T}$ according to the amount of *edition* required by the model $\mathcal{M}$ to induce the observations $\mathcal{T}$. We first define the two allowed *operations* to edit a given STRIPS action model $\mathcal{M}$:

- *Deletion*. A fluent $pre_f(\xi)/del_f(\xi)/add_f(\xi)$ is removed from the operator schema $\xi \in \mathcal{M}$, such that $f \in F_v(\xi)$.
- *Insertion*. A fluent $pre_f(\xi)/del_f(\xi)/add_f(\xi)$ is added to the operator schema $\xi \in \mathcal{M}$, s.t. $f \in F_v(\xi)$.

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 1. *Let $\mathcal{M}$ and $\mathcal{M}'$ be two* STRIPS *action models, such that they are* comparable. *The* **edit distance**, *denoted as $\delta(\mathcal{M}, \mathcal{M}')$, is the minimum number of* edit operations *that is required to transform $\mathcal{M}$ into $\mathcal{M}'$.*

Since F_v is a bound set, the maximum number of edits that can be introduced to a given action model defined within F_v is bound as well. In more detail, for an operator schema $\xi \in \mathcal{M}$ the maximum number of edits that can be introduced to their precondition set is $|F_v(\xi)|$ while the max number of edits that can be introduced to the effects is twice $|F_v(\xi)|$.

Definition 2. *The* ***maximum edit distance*** *of an* STRIPS *action model* $\mathcal{M}$ *built from the set of possible elements* F_v *is* $\delta(\mathcal{M}, *) = \sum_{\xi \in \mathcal{M}} 3 \times |F_v(\xi)|$.

We define now an edit distance to asses the mathing of a learned action model with respect to a plan trace $\mathcal{T}$.

Definition 3. *Given* $\mathcal{M}$*, a* STRIPS *action model built from* F_v*, and a plan trace* $\mathcal{T}$ *whose state observation are built with fluents in* F*. The* **observation edit distance***, denoted by* $\delta(\mathcal{M}, \mathcal{T})$*, is the minimal edit distance from* $\mathcal{M}$ *to any* comparable *model* $\mathcal{M}'$*, such that* $\mathcal{M}'$ *can produce a valid plan trace* $\mathcal{T}$*;*

$$\delta(\mathcal{M}, \mathcal{T}) = \min_{\forall \mathcal{M}' \rightarrow \mathcal{T}} \delta(\mathcal{M}, \mathcal{M}')$$

Note that the distance of an action model $\mathcal{M}$ with respect to a plan trace $\mathcal{T}$ could also be defined quantifying the amount of edition required by the observations of the plan execution to match the given model. This would imply defining *edit operations* that modify the fluents in the state observations instead of the *edit operations* that modify the action schemes (Sohrabi, Riabov, and Udrea 2016). Our definition of the *observation edit distance* is more practical since normally, F_v is smaller than F because the number of *variable objects* is smaller than the number of objects in the state observations.

The posterior probability distribution

According to the Bayes rule, the probability of an hypothesis $\mathcal{H}$ given the observations $\mathcal{O}$ can be computed with $P(\mathcal{H}|\mathcal{O}) = \frac{P(\mathcal{O}|\mathcal{H})P(\mathcal{H})}{P(\mathcal{O})}$. In our scenario, the hypotheses are about the set of possible STRIPS action models.

Given set of predicates Ψ and a given a set of operator headers (in other words, given the $F_v(\xi)$ sets) the size of the set of possible STRIPS models set is $\prod_\xi 2^{2|F_v(\xi)|}$, as explained in Section **??**. If we assume that a priori all models are equiprobable this means that $P(\mathcal{M}) = \frac{1}{\prod_\xi 2^{2|F_v(\xi)|}}$.

With respect to the observations, given Ψ and a set of objects Ω, the size of the possible state observations of length n, that is $\mathcal{O} = s_0, \ldots, s_n$ is given by $2^{n \times |F|}$.

With this regard, $P(\mathcal{M}|\mathcal{O})$, the probability distribution of the possible STRIPS models (within the $F_v(\xi)$ sets) given an observation sequence $\mathcal{O}$ could be computed by:

1. Computing the *observation edit distance* $\delta(\mathcal{M}, \mathcal{O})$ for every possible model $\mathcal{M}$ and mapping the *observation edit distance* into a likelihood with the following expression $1 - \frac{\delta(\mathcal{M}, \mathcal{O})}{\delta(\mathcal{M}, *)}$.
2. Applying the Bayes rule to obtain the normalized posterior probabilities, these probabilities must sum 1.

Computing the edit distance with classical planning

Our compilation is extensible to compute the *observation edit distance* by simply considering that the input STRIPS model $\mathcal{M}$, given in a learning task $\Lambda = \langle \mathcal{M}, \Psi, \mathcal{T} \rangle$, is *non-empty*. In other words, now $\mathcal{M}$ is a set of given operator schemas, wherein each $\xi \in \mathcal{M}$ initially contains $head(\xi)$ but also the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ sets. A solution to the planning task resulting from the extended compilation 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*. In theory, we could implement a third edit operation for *substituting* a fluent from a given operator schema. However, and with the aim of keeping a tractable branching factor of the planning instances that result from our compilations, we only implement *deletion* and *insertion*.
2. **Validates the edited model $\mathcal{M}'$ in the observed plan trace**. The solution plan continues with a postfix that validates the edited model $\mathcal{M}'$ on the given observations $\mathcal{T}$, as explained in Section **??** for the models that are programmed from scratch.

Now Λ does not formalize a learning task but the task of editing $\mathcal{M}$ to produce the plan trace $\mathcal{T}$, which results in the edited model $\mathcal{M}'$. The output of the extended compilation is a classical planning task $P'_\Lambda = \langle F_\Lambda, A'_\Lambda, I'_\Lambda, G_\Lambda \rangle$:

- F_Λ and G_Λ are defined as in the previous compilation.
- I'_Λ contains the fluents from F that encode s_0 and $mode_{prog}$ set to true. In addition, the input action model $\mathcal{M}$ is now encoded in the initial state. This means that the fluents $pre_f(\xi)/del_f(\xi)/add_f(\xi)$, $f \in F_v(\xi)$, hold in the initial state iff they appear in $\mathcal{M}$.
- A'_Λ, comprises the same three kinds of actions of A_Λ. The actions for *applying* an already programmed operator schema and the actions for *validating* an observation are defined exactly as in the previous compilation. The only difference here is that the actions for *programming* the operator schema now implement the two *edit operations* (i.e. include actions for *inserting* a precondition and for *deleting* a negative/positive effect).

To illustrate this, the plan of Figure 2 shows the plan for editing a given *blockswold* action model where again the positive effects `(handempty)` and `(clear ?v1)` of the `stack` schema are missing. In this case the edited action model is however validated at the plan shown in Figure **??**.

```
00 : (insert_add_handempty_stack)
01 : (insert_add_clear_stack_var1)
02 : (apply_unstack blockB blockA i1 i2)
03 : (apply_putdown blockB i2 i3)
04 : (apply_pickup blockA i3 i4)
05 : (apply_stack blockA blockB i4 i5)
06 : (validate_1)
```

Figure 2: Plan for editing a given *blockswold* schema and validating it at the plan shown in Figure **??**.

Our interest when computing the *observation edit distance* is not in the resulting action model $\mathcal{M}'$ but in the number of required *edit operations* (insertions and deleitions) for that $\mathcal{M}'$ is validated in the given observations, e.g. $\delta(\mathcal{M},\mathcal{T}) = 2$ for the example in Figure 2. In this case $\delta(\mathcal{M},*) = 3 \times 2 \times (11+5)$ since there are 4 action schemes (`pickup`, `putdown`, `stack` and `unstack`) and $|F_v| = |F_v(stack)| = |F_v(unstack)| = 11$ while $|F_v(pickup)| = |F_v(putdown)| = 5$ (as shown in Section **??**). The *observation edit distance* is exactly computed if the classical planning task resulting from our compilation is optimally solved (according to the number of edit actions); is approximated if it is solved with a satisfying planner; and is a less accurate estimate (but faster to be computed) if the solved task is a relaxation of the classical planning task that results from our compilation (Bonet and Geffner 2001).

Experiments

To evaluate the empirical performance of our method for *planning model recognition* we defined a set of possible STRIPS models, each representing a different *Turing Machines*, but all sharing the same tape alphabet and same set of machine states.

Modeling *Turing Machines* with STRIPS

A *Turing machine* is a tuple $M = \langle Q, q_o, Q_\perp, \mathcal{T}, \Box, \Sigma, \delta\rangle$:

- Q, is a finite and non-empty set of machine states such that $q_0 \in Q$ is the initial state of the machine and $Q_\perp \subseteq Q$ is the subset of acceptor states.
- $\mathcal{T}$ is the *tape alphabet*, that is a finite non-empty set of symbols that includes the *blank symbol* $\Box \in \mathcal{T}$ (the only symbol allowed to occur on the tape infinitely often) and that contains $\Sigma \subseteq \mathcal{T}$, the set of symbols allowed to initially appear in the tape (also called the *input alphabet*).
- $\delta : (Q \setminus Q_\perp) \times \mathcal{T} \rightarrow Q \times \{left, right\} \times \mathcal{T}$ is the *transition function*. If δ is not defined for the current pair of machine state and tape symbol, then the machine halts.

A table is the most common convention to represent the transitions defined by δ, where the table rows are indexed by the current tape symbol, while the table columns are indexed by the current machine state. For each possible pair of tape symbol and machine state, there is a table entry that defines: (1) the tape symbol to print at the current position of the header (2) whether the header is shifted *left* or *right* after the print operation and (3), the new state of the machine after the print operation. For instance, Figure 3 shows the table that represents the δ function of a *Turing Machine* for recognizing the $\{a^nb^nc^n : n \geq 1\}$ language. In this example the tape alphabet is $\Sigma = \{a, b, c, x, y, z, \Box\}$ while the possible 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.

A classical planning frame $\Phi = \langle F, A\rangle$ encodes the *transition function* δ of a *Turing Machine* M as follows. We assume that fluents F are instantiated from a set of *predicates* Ψ, as in PDDL (Fox and Long 2003). Each predicate $p \in \Psi$ has an argument list of arity $ar(p)$. Given a set of *objects* Ω that represent the cells in the tape of the given Turing Machine M, the set of fluents F is induced by assigning objects in Ω to the arguments of the predicates in Ψ; i.e. $F = \{p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}\}$, where Ω^k is the k-th Cartesian power of Ω. The predicates in Ψ are:

	q_0	q_1	q_2	q_3	q_4	$\underline{q_5}$
a	x,r,q_1	a,r,q_1	-	a,l,q_3	-	-
b	-	y,r,q_2	b,r,q_2	b,l,q_3	-	-
c	-	-	z,l,q_3	-	-	-
x	-	-	-	x,r,q_0	-	-
y	y,r,q_4	y,r,q_1	-	y,l,q_3	y,r,q_4	-
z	-	-	z,r,q_2	z,l,q_3	z,r,q_4	-
$\Box$	-	-	-	-	$\Box$,r,q_5	-

Figure 3: Example of a seven-symbol six-state *Turing Machine* for recognizing the $\{a^nb^nc^n : n \geq 1\}$ language ($\underline{q_5}$ is the only acceptor state).

- `(head ?x)` that encodes the current position of the header in the tape.
- `(next ?x1 ?x2)` encoding that the cell `?x2` follows cell `?x1` in the tape.
- `(symbol-`σ `?x)` encoding that the tape cell `?x` contains the symbol $\sigma \in \Sigma$.
- `(state-`q`)` encoding that $q \in Q$ is the current machine state.

Likewise we assume that actions $a \in A$ are instantiated from STRIPS operator schema. For each transition in δ, a STRIPS action schema is defined such that:

- The **header** of the schema is `transition-id(?xl ?x ?xr)` where *id* uniquely identifies the transition in δ and the parameters $?xl$, $?x$ and $?xr$ are tape cells.
- The **preconditions** of the schema includes `(head ?x)` and `(next ?xl ?x)` `(next ?x ?xr)` to force that $?x$ is the tape cell currently pointed by the header and that $?xl$ and $?xr$ respectively are its left and right neighbours. Additionally the schema includes preconditions `(symbol-`σ `?x)` and `(state-`q`)` to capture the symbol pointed by the header and the currrent machine state.
- The **delete effects** remove the symbol pointed by the header and the currrent machine state while the **positive effects** set the new symbol pointed by the header and the new machine state.

The STRIPS action schema of Figure 4 models the rule $a, q_0 \rightarrow x, r, q_1$ of the Turing Machine defined in Figure 3. The full encoding of the Turing Machine defined in Figure 3 produces a total of sixteen STRIPS action schema with the same structure as the one of Figure 4.

The execution of a *Turing Machine* can then be defined as a *plan trace* $\mathcal{T} = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ such that s_0 encodes the initial state of the tape plus the initial machine state and, for each $1 \leq i \leq n$, a_i is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$.

Related work

Conclussions

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

```
(:action transition-1 ;; a,$q_0 \rightarrow$ x,r,$q_1$
  :parameters (?xl ?x ?xr)
  :precondition (and (head ?x)
                     (symbol-a ?x) (state-$q_0$)
                     (next ?xl ?x) (next ?x ?xr))
  :effect (and (not (head ?x))
               (not (symbol-a ?x)) (not (state-$q_0$))
               (head ?xr) (symbol-x ?x) (state-$q_1$)))
```

Figure 4: STRIPS action schema that models the transition $a, q_0 \rightarrow x, r, q_1$ of the Turing Machine defined in Figure 3.

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.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language.

Pommerening, F.; Helmert, M.; and Bonet, B. 2017. Higher-dimensional potential heuristics for optimal classical planning.

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.

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.